The Forest

ZX Spectrum version, August 1983

This document is a collection of copies of the original design notes, source code print-outs, diagrams and publication informaion for the second published version of The Forest. It represents an archive of what was possible in a machine using an 8-bit microprocessor (Z80) and only 48 kilobytes of programmable memory.

My first personal computer was a TRS-80 Model 1 Level II and I have written separately (a document similar to this) about how the first version of The Forest was written for that and published in March 1983.

I then bought a Sinclair ZX Spectrum. It also used the Z80 processor, so much of my code was immediately transferable. However the Spectrum had a completely different way of displaying graphics, and in colour rather than monochrome. It also had a rather different flavour of BASIC for the higher level programming.

The machine had an interpreter for the programming language BASIC. That was stored in read-only memory (ROM) which was immediately available when power was switched on. Programs could either be typed into the remaining random-access memory (RAM) or be loaded from audio cassette tapes. One such program was an editor/assembler which made it possible to write additional programs in the Z80 machine code and save those on audio tape for reloading.

This version of The Forest was published by Phipps Associates, a company in South London.

The Forest was one of the earliest attempts to make a simulation of a sport: orienteering. That is, cross-country navigation on foot with map and compass to visit set control points in a planned order.

My particular innovation was to devise a way to generate limitless terrain from a very small amount of program code. The terrain was contoured and had several different types of vegetation plus towns and lakes. It had various kinds of point features dotted around, such as boulders and ponds.

Although I demonstrated that detailed terrain could be generated, the display hardware in those days was extremely limited. So a version of the program was made to output all of the map data as if surveying a real forest. Then the map could be drawn and printed just as we did for real orienteering events (I had been doing some of that since 1973). The published program was therefore accompanied by a full-colour printed map. This was seen as an advantage because it made it difficult to copy the product. People did not have colour printer/copiers in those days either.

The Z80 is fundamentally an 8-bit processor but with 16-bit memory addressing, allowing up to 64 kilobytes of memory. My Spectrum had the full 64kb, comprising 16kb of ROM plus 48kb of RAM. Some of the 8-bit registers could be programmed as pairs to facilitate 16-bit integer arithmetic.

For The Forest I had no access to any floating-point arithmetic so I had to make some interesting design choices in the first, TRS-80, version which I kept for the Spectrum version. For the (x, y) position of the orienteer on the map I used 3-byte fixed-point numbers: 2 bytes for the integer part and 1 byte after the decimal point. In my design notes it is written as HL.A to use the 16-bit HL register pair and the 8-bit accumulator. This enabled the map to have a width and height of 65km before repeating but the orienteer could be positioned to 1/256 of a metre. That was necessary to allow forward motion on any bearing, using sines and cosines. Which raised another point: how to do trigonometry without floating point? The answer involved 2 more design choices. Firstly compass bearings were “b-degrees” of which I had 256 to a complete circle: one byte’s worth. Bearings therefore had a resolution of 360 / 256 or about 1.4 degrees, more than accurate enough for an orienteer reading a compass. Then sines and cosines were looked up in a table of byte values, each being 128 times the true value and so represented as signed 1-byte integers. The table only needed 64 entries because values in the 4 quadrants are simply related. Also the table as read from the beginning gave sines but read backwards from the end it gave cosines. All to save memory and it worked very well.

The BASIC ROM had a floating-point calculator using 5-byte numbers (1 byte for the exponent). When I got the Melbourne House book of its full disassembly it was made clear that the calculator was not designed for speed. It would not have been fast enough for my purposes. Remember also that the clock speed was only about 4MHz.

The maps

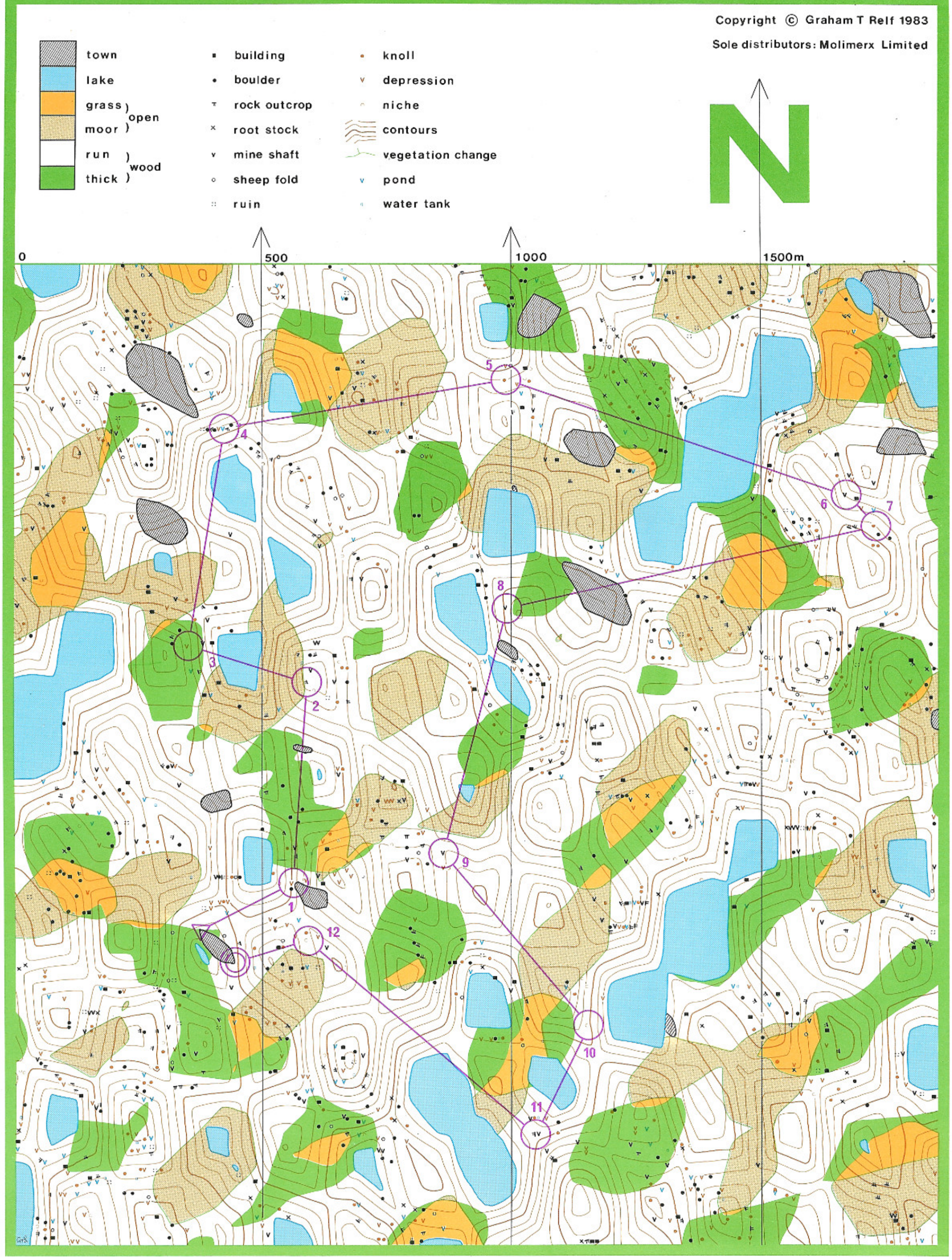
The next 2 pages show the printed orienteering maps which accompanied the published program. It has to be admitted that in the first one the contours are not very realistic. That was only the map issued for the TRS-80 version. While converting the program to run on the Spectrum I developed a much more realistic terrain generator which resulted in the second map, “The Complex Forest”. As will be shown in the listings this involved substituting a chunk of code in the terrain generator and providing a switching mechanism in the BASIC program. There are pages below which explain the difference in detail. (The same could be done retrospectively in the TRS-80 version if anyone is interested. I have discovered that there are enthusiasts still running those machines and resurrecting code such as mine.)

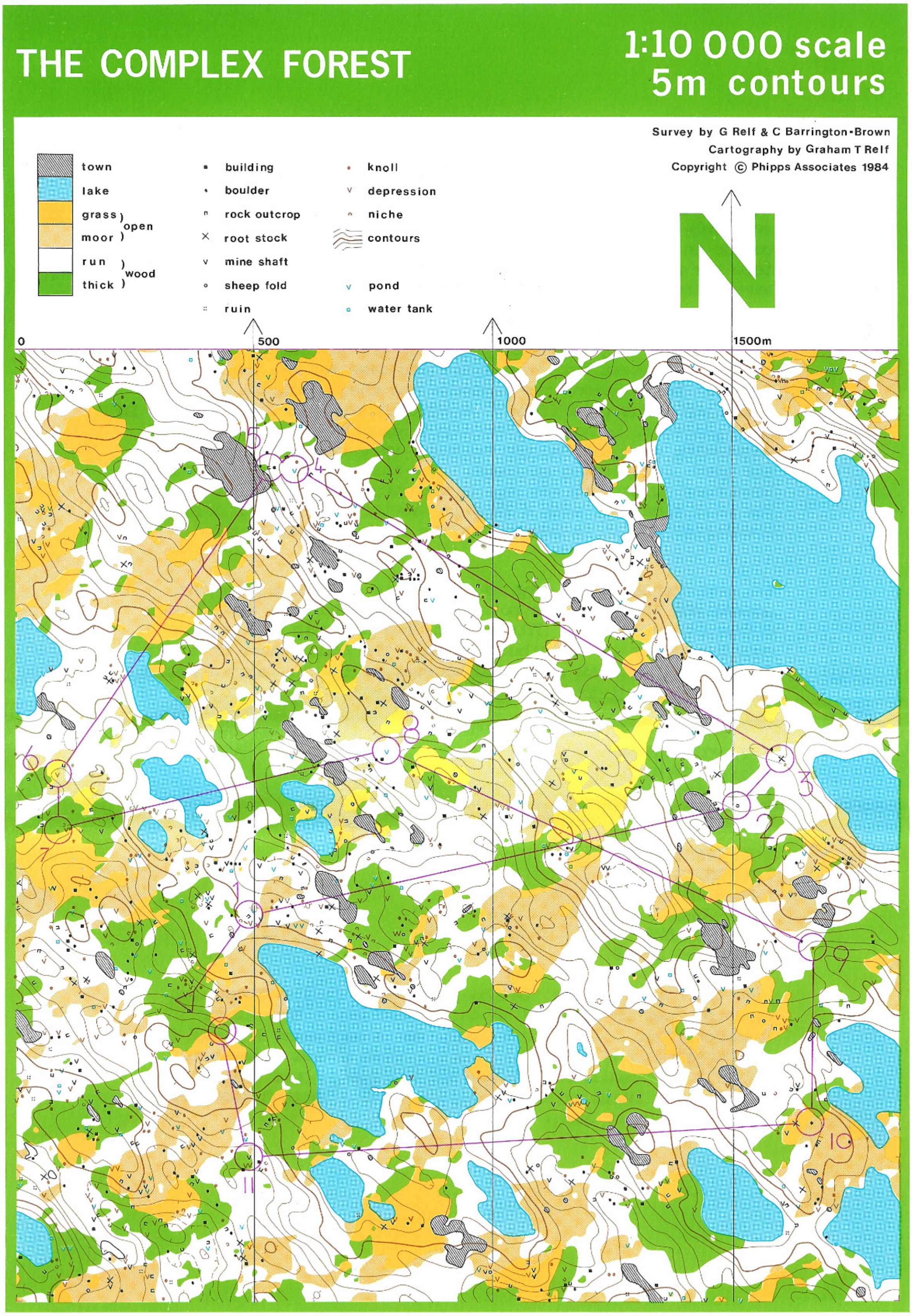
Graham Relf   August 2022

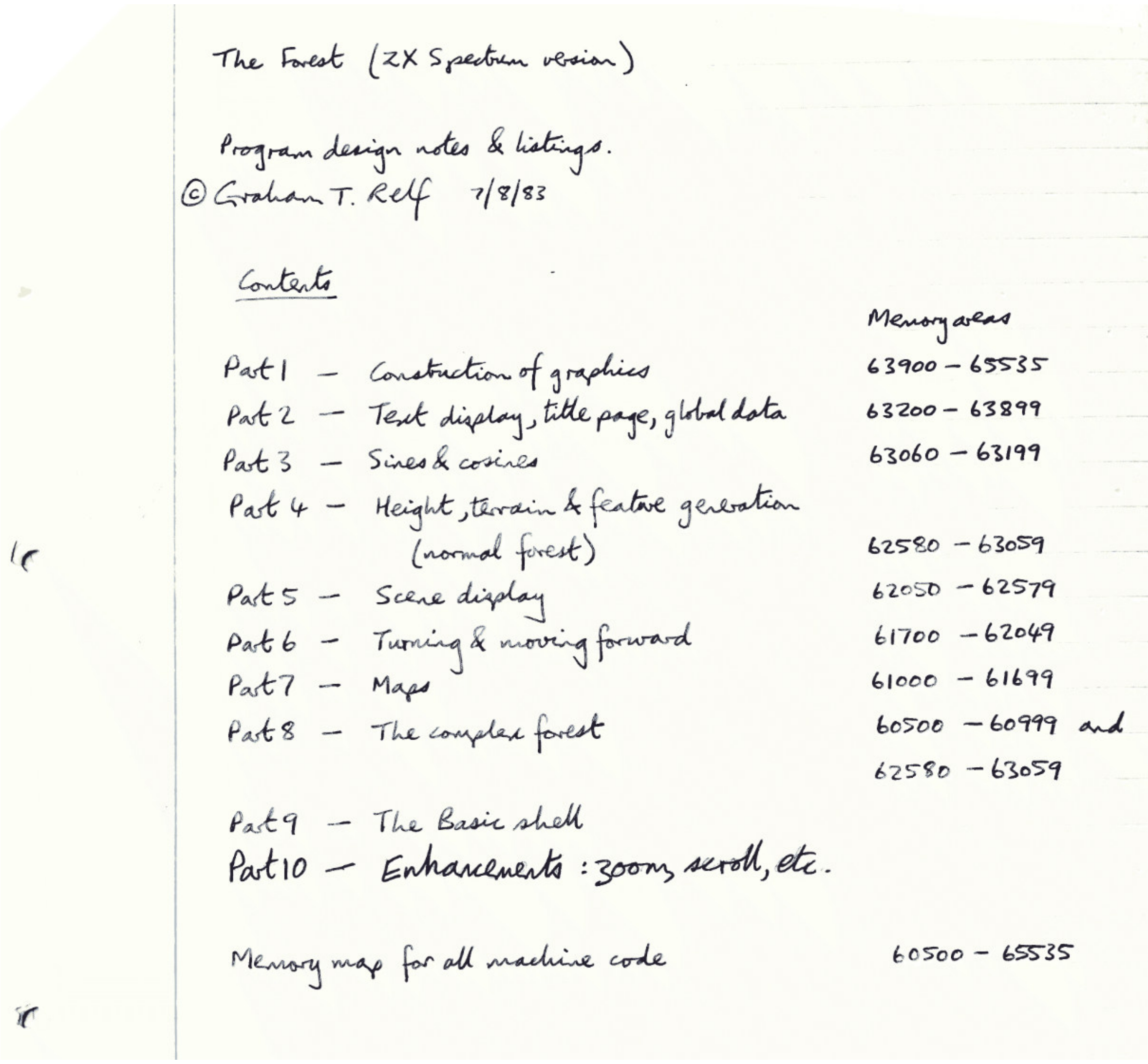
Page

THE FOREST

1:10 000 scale
5m contours




THE COMPLEX FOREST
1:10 000 scale
5m contours
Survey by G Relf & C Barrington-Brown
Cartography by Graham T Relf
Copyright © Phipps Associates 1984
town
lake
grass) open
moor)
run) wood
thick)
building
boulder
rock outcrop
root stock
mine shaft
sheep fold
ruin
knoll
depression
niche
contours
pond
water tank
N
0
500
1000
1500m



The Forest (ZX Spectrum version)

Program design notes & listings.

© Graham T. Relf 7/8/83

Contents

Many of the pages in this document are snapshots of my original handwritten notes.

But later you will see that I have provided most assembler routines as text, to facilitate possible reuse. If this typed-in code doesn't work I think the most likely cause will be that I have got addresses wrong in the various EQU lines. I have included some pages with memory maps which could be checked against the EQUs.

The Forest (ZX Spectrum version). Program design notes & listings.
Graham T. Relf 7/8/83

Part 1 — Construction of graphics.

In the displayed scene terrain types (except grass) and control flags are shown by graphical symbols. Each symbol is stored in memory as a 'graphic string' of special characters together with steering bytes. There is a machine code routine 'grafprint' which can display such a symbol anywhere on the screen, performing any necessary clipping, given the start address of the string and the starting address in the attributes part of the Spectrum screen map for the desired position.

The format of a graphic string is as follows.

For each character to be displayed:

byte no.	contents	
1	Attributes code	standard Sinclair meaning
2..9	Pixel bytes, from top down	
1Ø	Offset in attr. file to next display position. Offset Ø means end of graphic string.	

It is important to understand the memory mapping of the screen in the Spectrum. Binary pixel information is stored in a bit pattern in a display file starting at 4000H mapping 192 lines of 256 pixels each. Colour attributes are stored on a character basis (8×8 pixels each, 32 × 24 on the screen) one byte per character in an attribute file starting at 5800H. The attribute file maps the screen in a conventional fashion line by line from top left. The display file is much more complicated and is divided into three 64-line portions. The following two pages describe the layout of the display file.

Then follow the definitions of the special character sets used to construct the terrain symbols for The Forest, then a listing of program forgraf which compiles the graphics into graphic strings for display by the machine code routine which follows that.

ZX Spectrum

Order of storing 32-byte scan lines (0..191) in __display file__:

```
4000H  32  32  32
  ϕ, 8, 16, 24, 32, 40, 48, 56,  A        ⇐ Offsets to scroll up one char.
  32 32
   1, 9, 17, 25, ....          57,
    2, 10, ... etc.             58,
     3, 11, ....                 59,
      4, 12, ...                  60,
       5, 13, ...                  61,
        6, 14, ...                  62,
         7, 15, ...                  63,
4800H  32                    32
57×32 = 1824
A  64, 72, 80, 88, 96, 104, 112, 120,
    65, ...                       121,
     66, ...                       122,
      67, ...                       123,
       68, ...                       124,
        69, ...                       125,
         70, ...                       126,
          71, ...                       127,
5000H  32                          32
1824  128, 136, 144, 152, 160, 168, 176, 184,
       129, ...                         185,
        130, ...                         186,
         131, ...                         187,
          132, ...                         188,
           133, ...                         189,
            134, ...                         190,
             135, ...                         191.
```

5800H

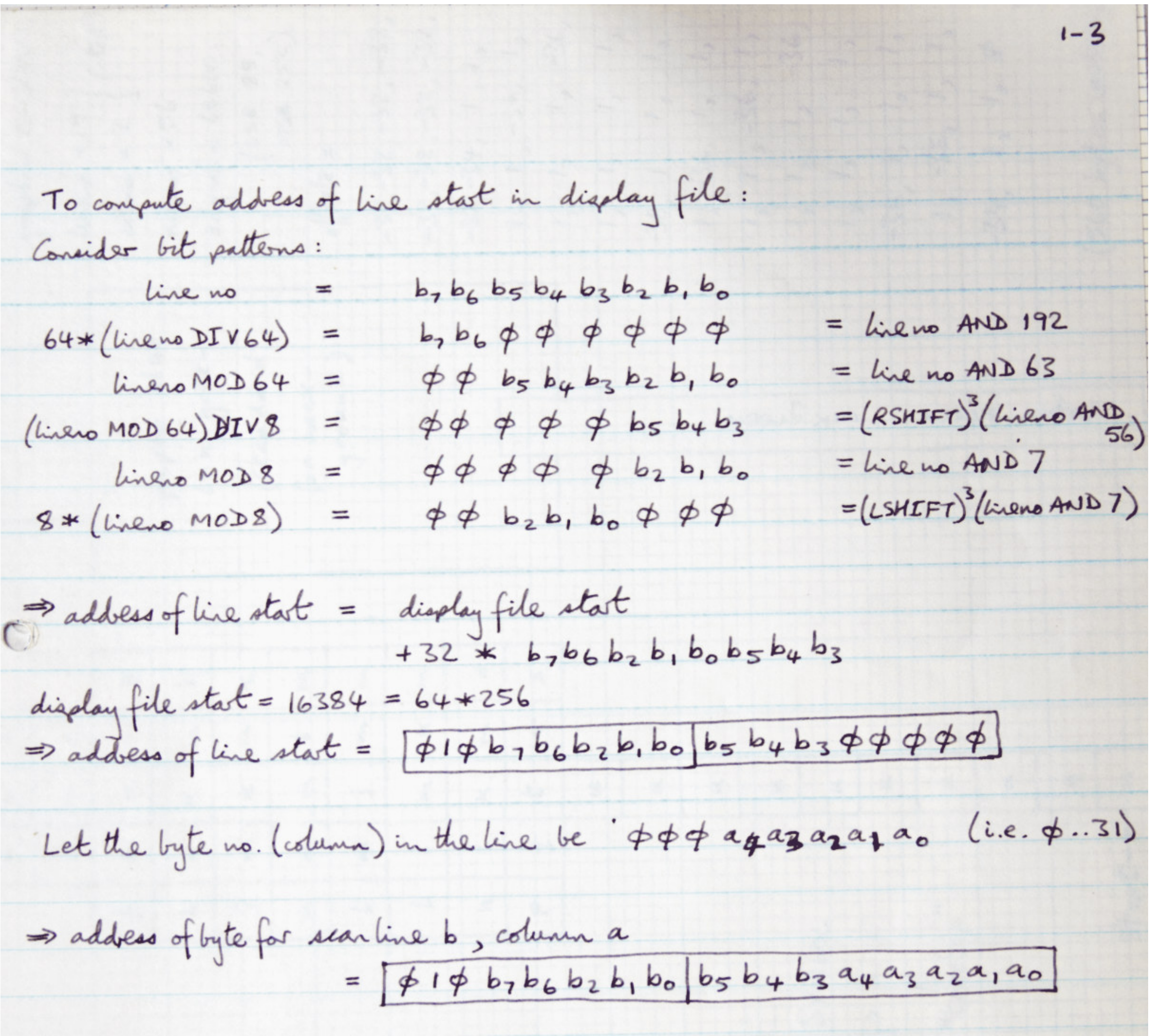
⇒ address of line start in display file = display file start

+32 * [64 * (line no DIV 64)

+ 8 * (line no MOD 8)

+ (line no MOD 64) DIV 8]

: line no = ϕ..191, scan lines from top of screen downwards.



1-3

To compute address of line start in display file:

Consider bit patterns:

line no	=	$b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$	
64*(lineno DIV64)	=	$b_7 b_6 \phi \phi \phi \phi \phi \phi$	= lineno AND 192
lineno MOD 64	=	$\phi \phi b_5 b_4 b_3 b_2 b_1 b_0$	= line no AND 63
(lineno MOD 64)DIV 8	=	$\phi \phi \phi \phi \phi b_5 b_4 b_3$	$=(\text{RSHIFT})^3(\text{lineno AND } 56)$
lineno MOD 8	=	$\phi \phi \phi \phi \phi b_2 b_1 b_0$	= line no AND 7
8*(lineno MOD8)	=	$\phi \phi b_2 b_1 b_0 \phi \phi \phi$	$=(\text{LSHIFT})^3(\text{lineno AND } 7)$

⇒ address of line start = display file start
+32 * $b_7 b_6 b_2 b_1 b_0 b_5 b_4 b_3$

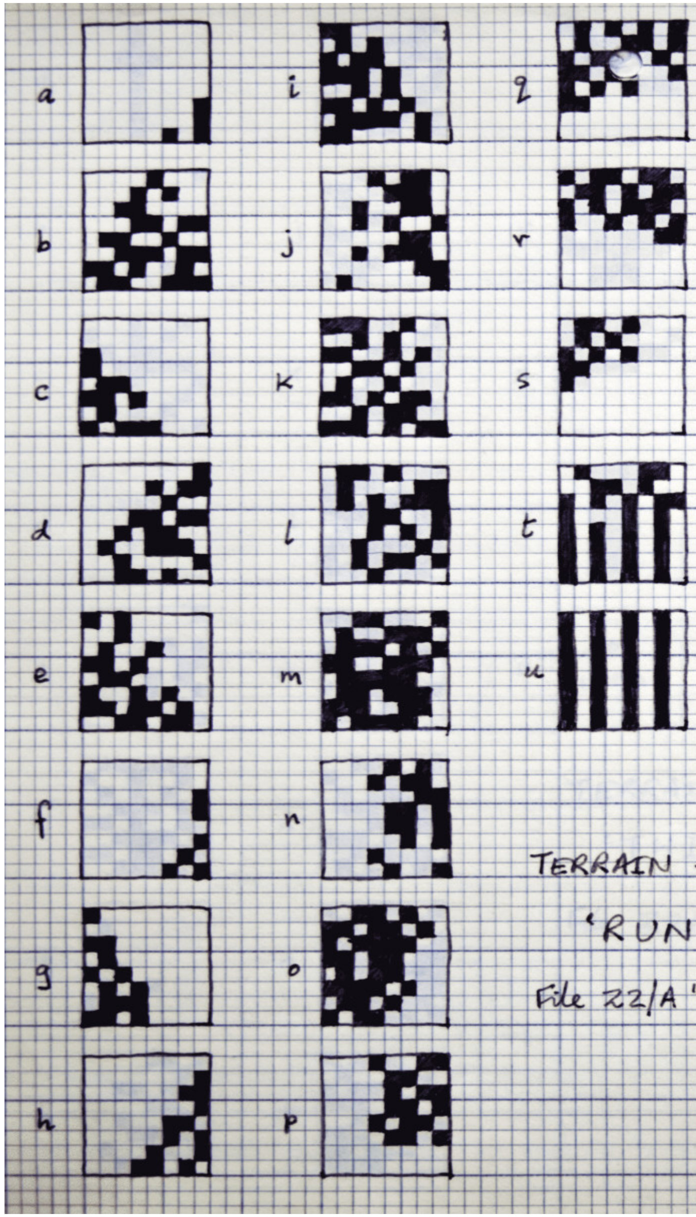
display file start = 16384 = 64*256

⇒ address of line start = $\boxed{\phi 1 \phi b_7 b_6 b_2 b_1 b_0 \mid b_5 b_4 b_3 \phi \phi \phi \phi \phi}$

Let the byte no. (column) in the line be $\phi \phi \phi a_4 a_3 a_2 a_1 a_0$ (i.e. ϕ..31)

⇒ address of byte for scan line b, column a

$= \boxed{\phi 1 \phi b_7 b_6 b_2 b_1 b_0 \mid b_5 b_4 b_3 a_4 a_3 a_2 a_1 a_0}$

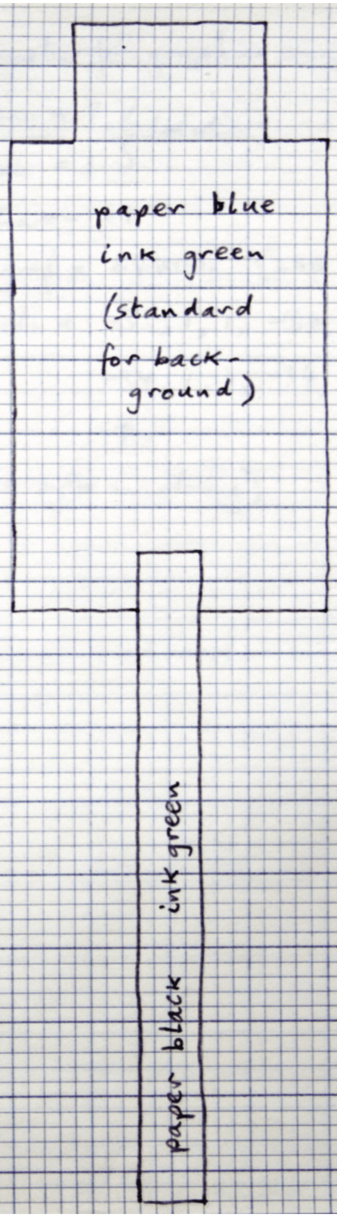


TERRAIN SYMBOL

'RUN'

File 22/A "treetop"

		a	b	c	
		d	m	e	
	f	m	k	m	g
	h	k	k	m	i
	j	m	k	m	k
	m	m	m	l	m
	l	m	j	m	m
	l	l	m	k	m
	n	l	k	m	o
	p	q	t	r	s
			u		
			u		
			u		
			u		
			u		
			u		
			u		
			u		
			u		
Start →			u		


paper blue
ink green
(standard
for back-
ground)

paper black ink green

Parameters for
graphics compiler:

lineno = 19 } (start)
colno = 2 }

nchars = 56

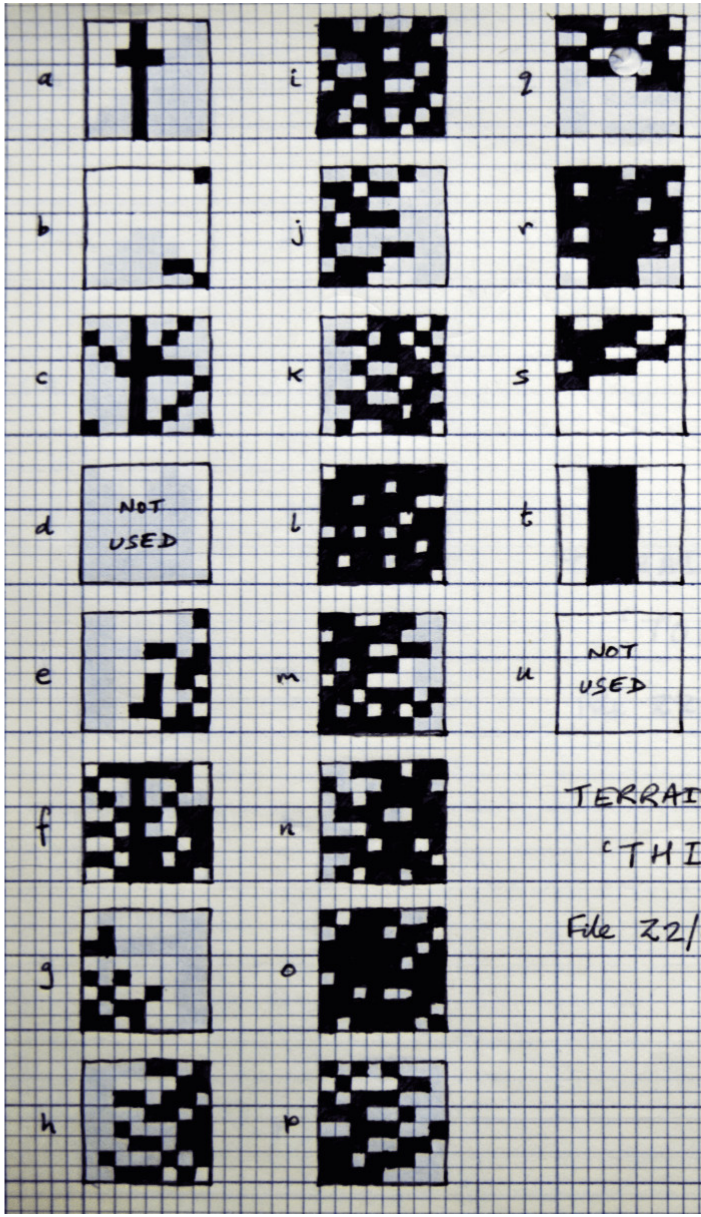
adgraf = 64600.
(LSB 88,
MSB 252)

offsets =

-32,-32,-32,-32,
-32,-32,-32,-32,
-32,-34, 1, 1,
1, 1,-36, 1,
1, 1, 1, -36,
1, 1, 1, 1,
-36, 1, 1, 1,
1, -36, 1, 1,
1, 1,-36, 1,
1, 1, 1, -36,
1, 1, 1, 1,
-36, 1, 1, 1,
1, -35, 1, 1,
-34, 1, 1, ϕ.

I4

(560 bytes in string)



		a	
	b	c	
	e	f	g
	h	i	j
	k	l	m
	n	o	p
	q	r	s
Start →		t	

Attributes all standard: paper blue, ink green.

TERRAIN SYMBOL 'THICK'

File Z2/A "fir"

Parameters for graphics compiler:

hieno = 7 } (start)
colno = 1 }

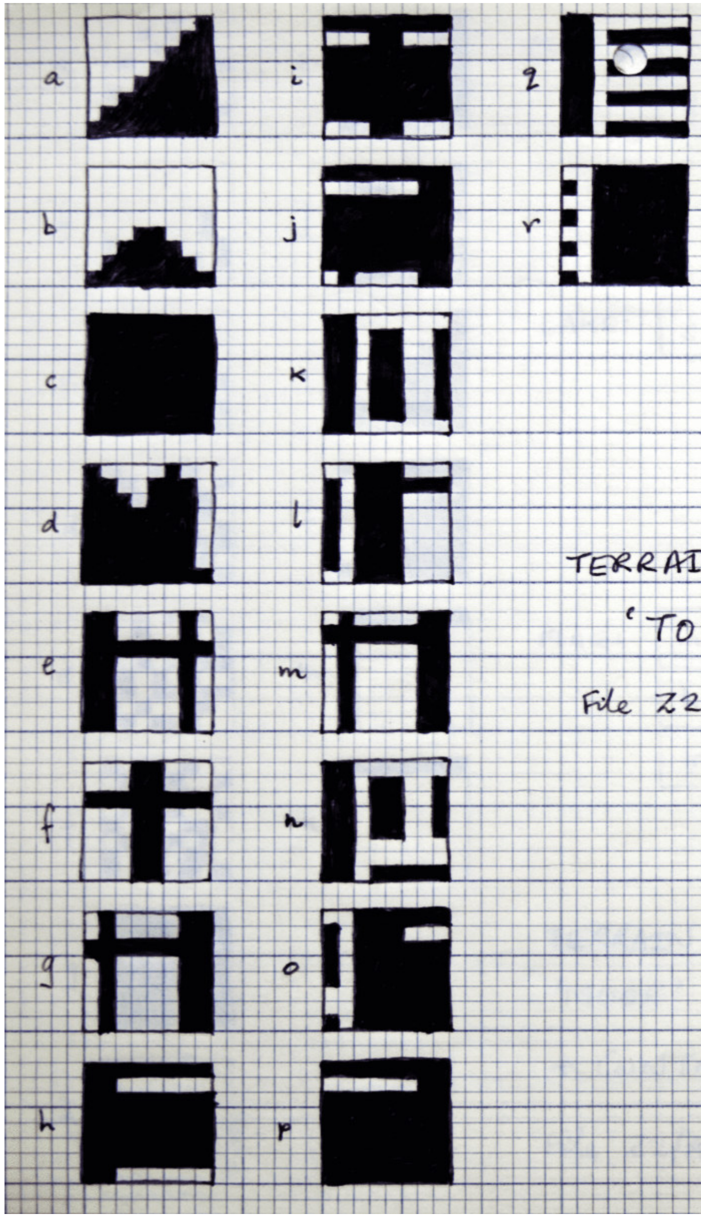
nchars = 19

adgraf = 64400. (LSB 144, MSB 251)

offsets = −33, 1, 1, −34, 1, 1, −34, 1, 1,
−34, 1, 1, −34, 1, 1, −34, 1, −32,
ϕ.

(190 bytes in string)

1-5



	b	
a	c	d
e	f	g
h	i	j
k	l	m
n	o	p
q	r	c

↑
Start

Paper blue, ink magenta

Paper white, ink magenta

TERRAIN SYMBOL
'TOWN'

File Z2/A "house"

Parameters for graphics compiler:

lineno = 6 } (start)
colno = 1 }

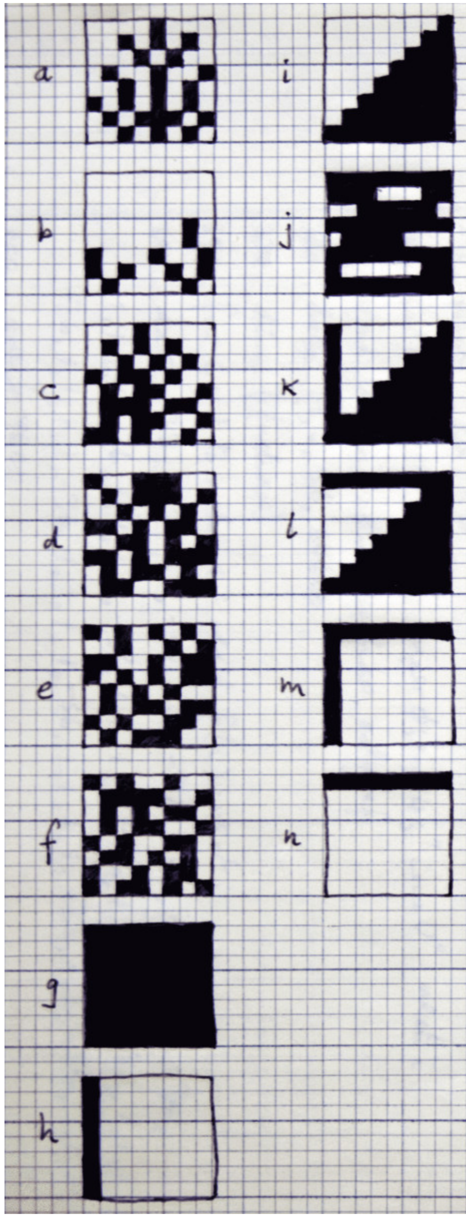
nchars = 19

adgraf = 64200 (LSB 200, MSB 250)

offsets = 1, -2, -32, 1, 1, -34, 1, 1, -34,
1, 1, -34, 1, 1, -34, 1, 1, -33,
∅.

(190 bytes in string)

1-6



MOOR:

a	b	c
d	e	f

↑
Start

Paper blue,
ink yellow

lineno = 1
colno = 1
nchars = 6
adgraf = 64100 (LSB 100, MSB 250)
offsets = 1, -32, -1, -1, 32, ϕ.
(60 bytes in string)

LAKE:

j	j	j

↑
Start

Paper blue,
ink cyan

lineno = ϕ
colno = 1
nchars = 3
adgraf = 63900 (LSB 156, MSB 249)
offsets = 1, -2, ϕ.
(30 bytes in string)

FLAG:

m	n	l
h	i	g
k	g	g

↑
Start

Paper white,
ink red

lineno = 2
colno = 1
nchars = 9
adgraf = 64000 (LSB 0, MSB 250)
offsets = 1, -2, -32, 1, 1, -34, 1, 1, ϕ.

TERRAIN SYMBOLS
'MOOR' & 'LAKE'
+ CONTROL FLAG

File Z2/A "moor"

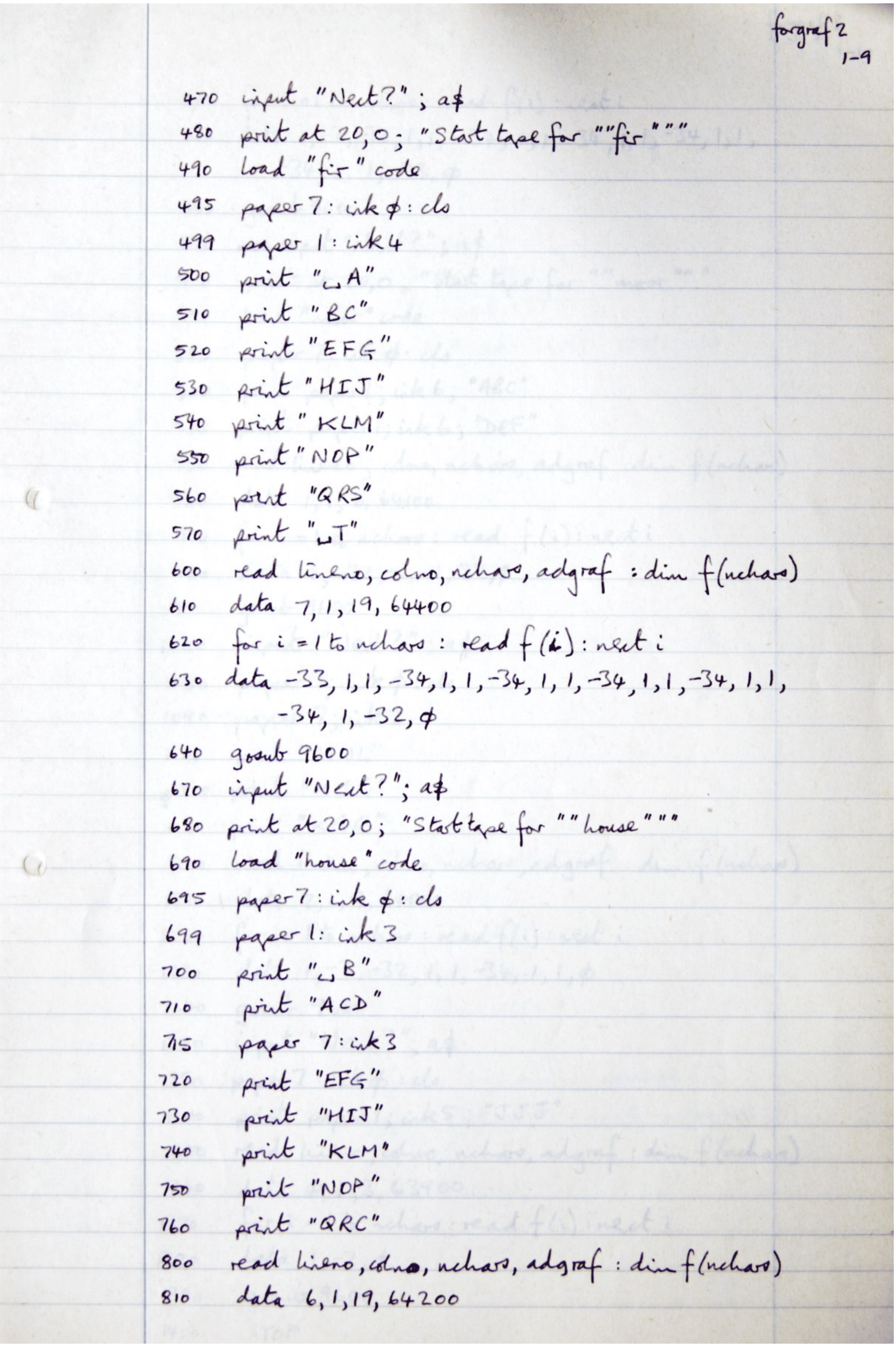
1-7

Program "forgraf" : Forest graphics compiler.
(A nasty little program, but it only had to work once!)
For each graphic :
  load special character set,
  display graphic in top left corner of screen,
  read parameters for compiling graphic,
  compile the graphic into a string in memory.

NB: protect the memory to be used for the graphic strings before running this program.

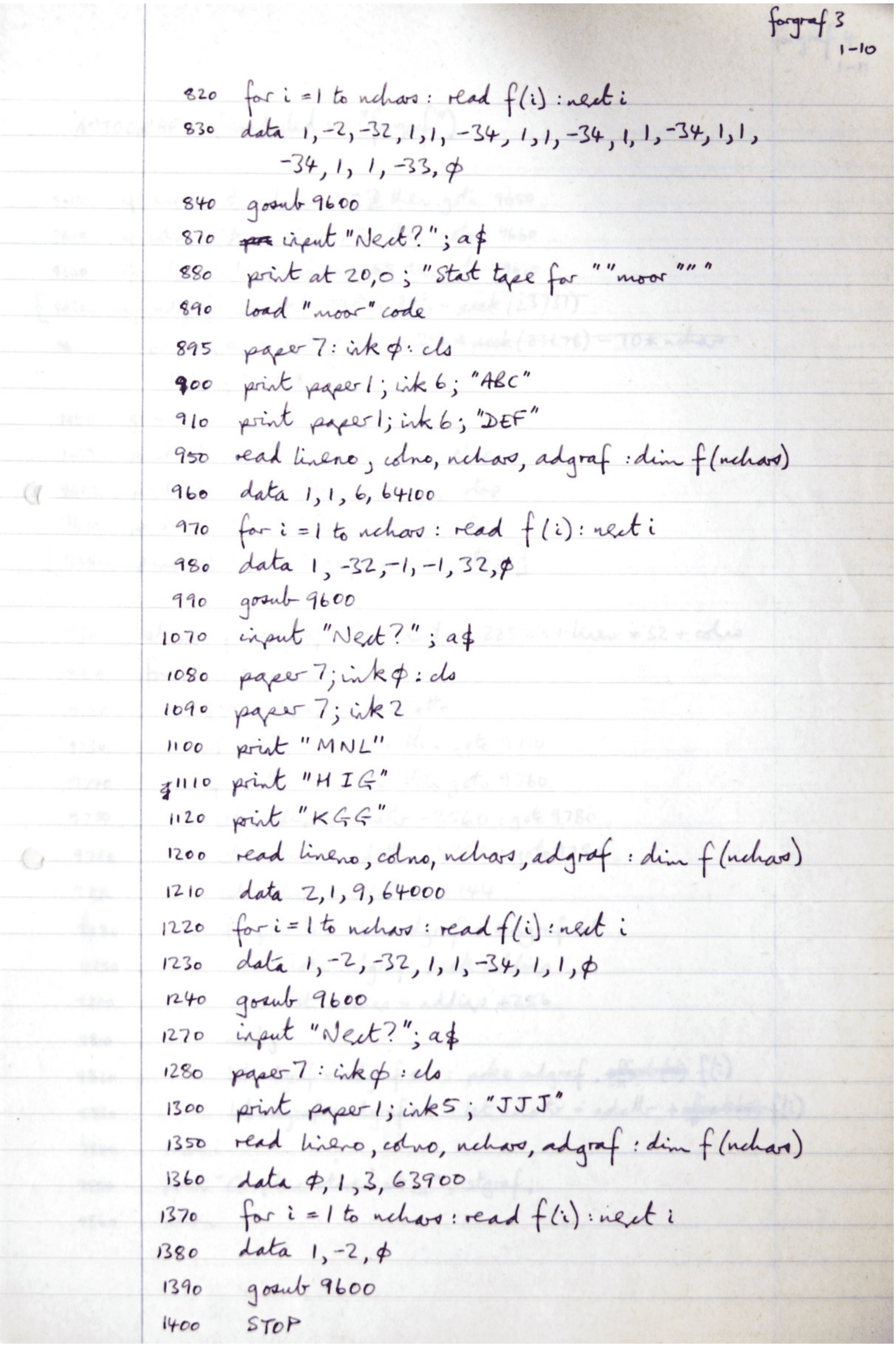
```
180  print "Start tape for ""treetop"""
190  load "treetop" code
195  paper 7: ink ϕ : cls
199  paper 1 : ink 4
200  print "␣ABC"
210  print "␣DME"
220  print "FMKMG"
230  print "HKKMI"
240  print "JMKMK"
250  print "MMMLM"
260  print "LMJ MM"
270  print "LLMKM"
280  print "NLKMO"
290  print "PQTRS"
300  for i=1 to 10 : print paper ϕ; ink4;"␣␣U" : next i
350  read lineno, colno, nchars, adgraf
360  data 19,2,56,64600
370  dim f(nchars)
380  for i=1 to nchars : read f(i):next i
390  data -32,-32,-32,-32,-32,-32,-32,-32,-32,-34,
     1,1,1,1,-36,1,1,1,1,-36,1,1,1,1,-36,1,1,1,1,-36,1,1,1,1,
    -36,1,1,1,1,-36,1,1,1,1,-36,1,1,1,1,-35,1,1,-34,1,1,ϕ
400  gosub 9600
```

NB: All these prints have graphics shifts just inside the quotes. The upper case letters here are the ROM-loaded defaults for the ~~g~~ user-definable characters.



forgraf2
1-9

```
470 input "Next?"; a$
480 print at 20,0; "Start tape for ""fir"""
490 load "fir" code
495 paper 7: ink ∅: cls
499 paper 1: ink 4
500 print "␣A"
510 print "BC"
520 print "EFG"
530 print "HIJ"
540 print "KLM"
550 print "NOP"
560 print "QRS"
570 print "␣T"
600 read lineno, colno, nchars, adgraf : dim f(nchars)
610 data 7,1,19,64400
620 for i=1 to nchars : read f(i): next i
630 data -33,1,1,-34,1,1,-34,1,1,-34,1,1,-34,1,1,
        -34,1,-32,∅
640 gosub 9600
670 input "Next?"; a$
680 print at 20,0; "Start tape for ""house"""
690 load "house" code
695 paper 7: ink ∅: cls
699 paper 1: ink 3
700 print "␣B"
710 print "ACD"
715 paper 7: ink 3
720 print "EFG"
730 print "HIJ"
740 print "KLM"
750 print "NOP"
760 print "QRC"
800 read lineno, colno, nchars, adgraf : dim f(nchars)
810 data 6,1,19,64200
```



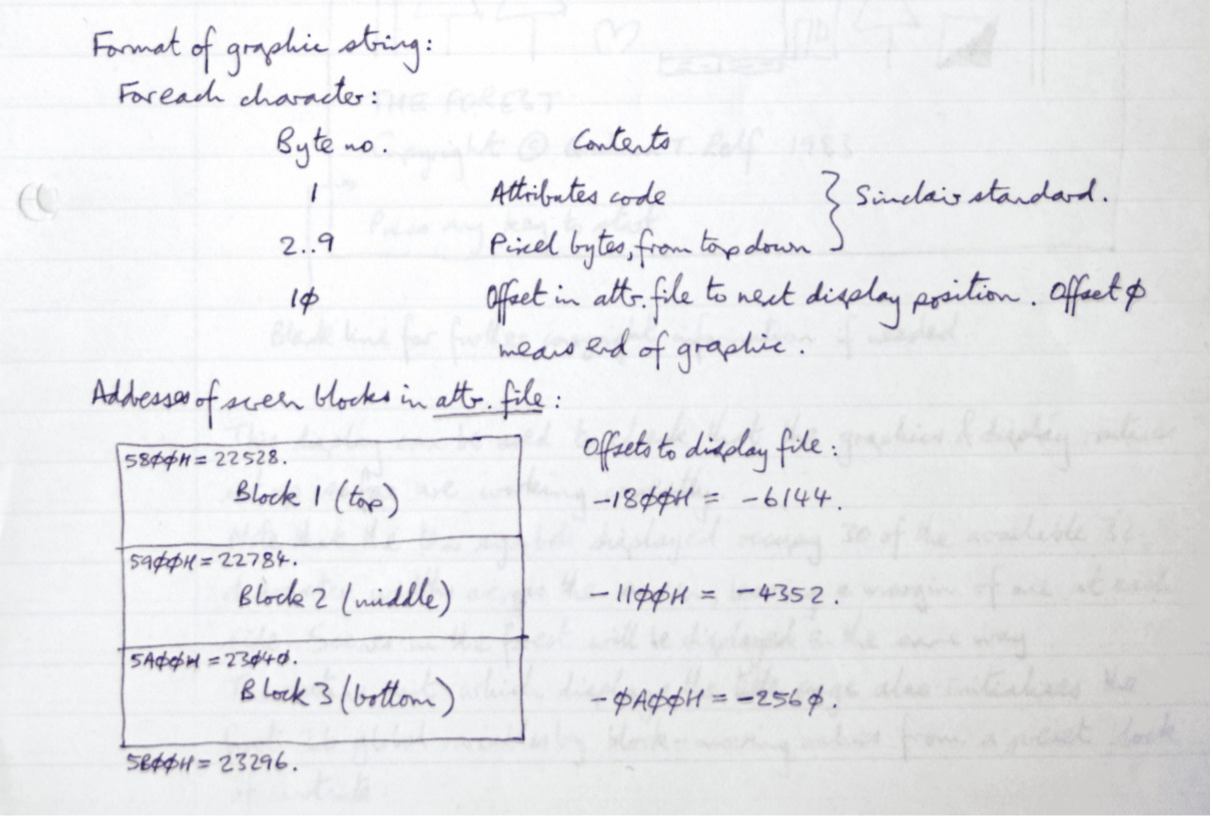
forgraf 3
1-10

```
820  for i=1 to nchars: read f(i):next i
830  data 1,-2,-32,1,1,-34,1,1,-34,1,1,-34,1,1,
        -34,1,1,-33,ϕ
840  gosub 9600
870  ~~pri~~ input "Next?";a$
880  print at 20,0;"Start tape for ""moor"""
890  load "moor" code
895  paper 7: ink ϕ: cls
900  print paper 1; ink 6; "ABC"
910  print paper 1; ink 6; "DEF"
950  read lineno, colno, nchars, adgraf :dim f(nchars)
960  data 1,1,6,64100
970  for i=1 to nchars: read f(i): next i
980  data 1,-32,-1,-1,32,ϕ
990  gosub 9600
1070 input "Next?"; a$
1080 paper 7; ink ϕ : cls
1090 paper 7; ink 2
1100 print "MNL"
1110 print "HIG"
1120 print "KGG"
1200 read lineno,colno,nchars,adgraf : dim f(nchars)
1210 data 2,1,9,64000
1220 for i=1 to nchars: read f(i): next i
1230 data 1,-2,-32,1,1,-34,1,1,ϕ
1240 gosub 9600
1270 input "Next?"; a$
1280 paper 7 : ink ϕ : cls
1300 print paper 1; ink 5; "JJJ"
1350 read lineno, colno, nchars, adgraf : dim f(nchars)
1360 data ϕ,1,3,63900
1370 for i=1 to nchars: read f(i): next i
1380 data 1,-2,ϕ
1390 gosub 9600
1400 STOP
```

AUTOGRAF (included in "forgraf")

```
9600  if lineno < ϕ or lineno > 22 then goto 9650
9610  if colno < ϕ or colno > 31 then goto 9660
9620  if nchars < 1 ~~or nchars > 255~~ then goto 9670
~~[9630  if adgraf < peek (23730) + 256 * peek (23731)~~
~~          or adgraf > peek (23675) + 256 * peek (23676) - 10 * nchars~~
~~           then goto 9680 ]~~
9640  goto 9700
9650  print at 21,ϕ; "Line no. error"; : stop
9660  print at 21,ϕ; "Column no. error"; : stop
9670  print at 21,ϕ; "Nchars error"; : stop
~~[9680  print at 21,ϕ; "Adgraf error"; : stop ]~~

9700  let stgraf = adgraf : let adattr = 22528 + lineno * 32 + colno
9710  for i = 1 to nchars
9720        poke adgraf, peek adattr
9730        if adattr < 22784 then goto 9770
9740        if adattr < 23040 then goto 9760
9750        let addisp = adattr - 2560 : goto 9780
9760        let addisp = adattr - 4352 : goto 9780
9770        let addisp = adattr - 6144
9780        for j = 1 to 8 : let adgraf = adgraf + 1
9790              poke adgraf, peek addisp
9800              let addisp = addisp + 256
9810        next j
9820        let adgraf = adgraf + 1 : poke adgraf, ~~offset(i)~~ f(i)
9830        let adgraf = adgraf + 1 : let adattr = adattr + ~~offset(i)~~ f(i)
9840  next i
9850  print "Graphic stored at ␣"; stgraf;
9860  return
```

```
                       ; New GRAFPRINT
       ORG 65200       ; Position just below UDG. (User-Defined Graphics)
ADATT: DEFW 0          ; 65200 Pointer to display position in attr file
                       ; (22528..23295)
ADGRA: DEFW 0          ; 65202 Pointer to first byte of graphics string
GSHOW: LD   DE,(ADGRA) ; 65204 Point to first byte of graphic
       LD   HL,(ADATT) ; 65211 Point to display position in attr file
G0:    PUSH HL         ; Save attr file address
       LD   A,H        ; Look at MSB:
       CP   88         ; If < 88
       JR   C,OFFSC    ; then off top of screen
       CP   89         ; else if < 89
       JR   C,BLK1     ; then in top block of screen
       CP   90         ; else if < 90
       JR   C,BLK2     ; then in middle block of screen
       CP   91         ; else if >= 91
       JR   NC,OFFSC   ; then off bottom of screen
BLK3:  LD   BC,2560    ; Offset from bottom attr block to display file
       JR   COMMN      ; Join common path
BLK2:  LD   BC,4352    ; Offset from middle attr block to display file
       JR   COMMN      ; Join common path
BLK1:  LD   BC,6144    ; Offset from top attr block to display file
COMMN: LD   A,(DE)     ; Get 1st byte of character, the attribute code
       LD   (HL),A     ; Set attributes for this character
       AND  A          ; Clear carry
       SBC  HL,BC      ; Point to corresponding position in display file
       INC  DE         ; Point to first pixel byte this character
       LD   B,8        ; Count 8 pixel bytes
GLOOP: LD   A,(DE)     ; Get pixel byte and
       LD   (HL),A     ; display it
       DEC  BC         ; If 8 bytes done for this character
       JR   Z,EXITL    ; then exit loop
       INC  H          ; Point to next display byte this char, 256 forward
       INC  DE         ; Point to next pixel byte in character
       JR   GLOOP      ; Next pixel byte
OFFSC: LD   B,8        ; Off screen: loop to skip to 8th pixel this char
OLOOP: INC  DE
       DJNZ OLOOP
EXITL: POP  HL         ; Restore pointer to attr file
       INC  DE         ; Point to last byte this char, the offset to next
in attr file
       LD   A,(DE)     ; Get offset
       CP   0          ; If zero
       RET  Z          ; then end of string
       LD   B,0        ; Clear B
       CP   128        ; If offset < 0
       JR   C,POSOK
       DEC  B          ; then B = 255, ie extend sign for BC
POSOK: LD   C,A        ; BC = (signed) offset
       ADD  HL,BC      ; Add offset to attr file ptr to get new display posn
       INC  DE         ; Point to 1st byte of next character
       JR   G0         ; Do next character
```



Format of graphic string:

For each character:

Byte no.	Contents	
1	Attributes code	} Sinclair standard.
2..9	Pixel bytes, from top down	
1Ø	Offset in attr. file to next display position. Offset Ø means end of graphic.	

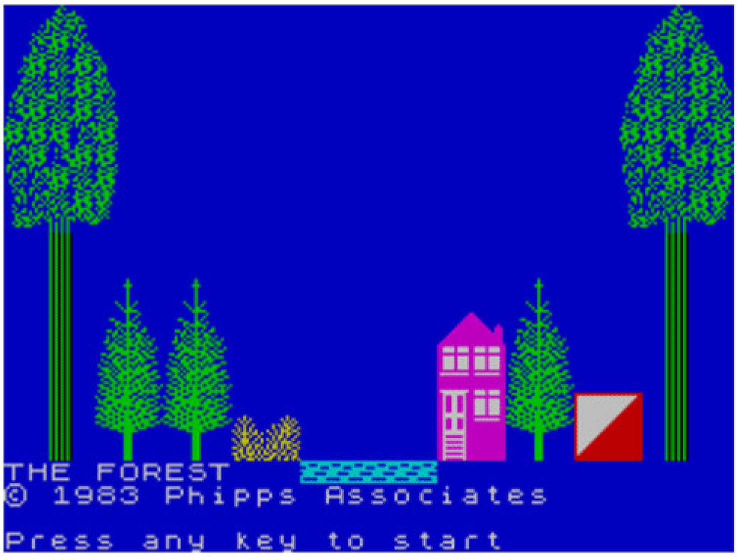
Addresses of screen blocks in attr. file:

		Offsets to display file:
58ØØH = 22528.	Block 1 (top)	−18ØØH = −6144.
59ØØH = 22784.	Block 2 (middle)	−11ØØH = −4352.
5AØØH = 23Ø4Ø.	Block 3 (bottom)	−ØAØØH = −256Ø.
5BØØH = 23296.		

Part 2 — Text string display and the title page, global data.

The next routine, textprint, displays a normal character string anywhere on screen (in the 32×24 character grid) given the starting address of the string and its required position in the attr. file. The character strings are held in memory as a sequence of ASCII bytes ending with zero (null).

The title page, displayed after The Forest has been loaded from tape shows examples of the terrain symbols (except 'grass', which is displayed differently) together with text strings for the title, copyright notice & a prompt for continuing.


THE FOREST
© 1983 Phipps Associates

Press any key to start

Blank line for further copyright information if needed.

This display can be used to check that the graphics & display routines set up so far are working correctly.

Note that the ten symbols displayed occupy 30 of the available 32 character widths across the screen, leaving a margin of one at each side. Scenes in the forest will be displayed in the same way.

The routine init which displays the title page also initialises the first 26 global variables by block-moving values from a preset block of constants.

```
       ORG 63850    ; CLEAR SCREEN
SDF:   EQU 16384    ; Start of display file
SAF:   EQU 22528    ; Start of attribute file
ATTR0: DEFB 0       ; Attributes byte for cleared screen: poke a value
ZERO:  DEFB 0       ; Value for display file
CLS:   LD BC,32*24  ; 24 lines to set in attr file
       LD DE,SAF    ; Destination for move
       LD HL,ATTR0  ; Source for move
       CALL CLOOP   ; Move
       LD BC,32*192 ; 192 lines to clear in display file
       LD DE,SDF    ; Destination for move
       LD HL,ZERO   ; Source for move
CLOOP: LDI          ; Move
       DEC HL       ; Keep pointing to single source byte
       JP PE,CLOOP  ; Repeat until BC = 0
       RET


       ORG 63745     ; TEXTPRINT
ATTR:  DEFB 0        ; Attributes to be used for string
ADATT: DEFW 0        ; Pointer to display position in attr file
ADTEX: DEFW 0        ; Pointer to 1st byte of text string
TSHOW: LD DE,(ADTEX) ; Point to string
       LD HL,(ADATT) ; Point to attr file
T0:    PUSH HL       ; Save attr file pointer
       LD A,H        ; Look at MSB
       CP 88         ; If < 88
       JR C,OFFSC    ; then off top of screen
       CP 89         ; else if < 89
       JR C,BLK1     ; then in top block
       CP 90         ; else if < 90
       JR C,BLK2     ; then in middle block
       CP 91         ; else if >= 91
       JR NC,OFFSC   ; then off bottom of screen
       LD BC,2560    ; Offset to display file, bottom block
       JR COMMN
BLK2:  LD BC,4352    ; Offset to display file, middle block
       JR COMMN
BLK1:  LD BC,6144    ; Offset to display file, top block
COMMN: LD A,(DE)     ; Get ASCII byte from string
       CP 0          ; If null
       JR Z,ENDT     ; then end of string
       LD A,(ATTR)   ; Get attribute byte for string
       LD (HL),A     ; Set attributes for this character
       AND A         ; Clear carry
       SBC HL,BC     ; Offset pointer to display file
       PUSH DE       ; Save string pointer
       EX DE,HL      ; DE = display file pointer
       LD L,(HL)     ; Get ASCII code byte
       LD H,0        ; HL = ASCII code byte
       ADD HL,HL
       ADD HL,HL
       ADD HL,HL      ; HL = 8 * ASCII code byte
       LD BC,15360    ; Base addr of std char set in ROM (=15616 - 32 * 8)
       ADD HL,BC      ; HL points to 1st pixel byte in ROM for ASCII char
       LD B,8         ; Count 8 pixel bytes
TLOOP: LD A,(HL)      ; Get pixel byte
       LD (DE),A      ; Display it
       DEC B          ; If 8 bytes displayed
       JR Z,EXITL     ; then character done
       INC D          ; Point to next display file position this char, 256
forward
       INC HL         ; Point to next pixel byte
       JR TLOOP       ; Next byte
EXITL: POP DE         ; Restore string pointer
OFFSC: INC DE         ; Next char in string
       POP HL         ; Restore attr file pointer
       INC HL         ; Next attr file display position
       JR T0          ; Next character
ENDT:  POP HL         ; Tidy the stack
       RET
```

Format for stored text string: 1 byte ASCII code for each character followed by zero byte to show end of string.

```
       ORG 63200      ; INIT (Title page & initialise global data)
T0:    EQU 63757
G0:    EQU 65210
CLS:   EQU 63852
ATTR0: EQU 63850
TATTR: EQU 63745
TITLE: DEFM "The Forest"
       DEFB 0
COPYR: DEFM "Copyright © Graham T Relf 1983   " ; 32 characters
DISTR: DEFS 32,32                    ; Spare line for distributor name
PKEY:  DEFM "Press any key to start"
       DEFB 0
INIT:  LD A,12        ; Paper blue, ink green
       LD (ATTR0),A
       CALL CLS
       LD A,15        ; Paper blue, ink white
       LD (TATTR),A
       LD DE,TITLE
       LD HL,23168    ; Line 20, column 0
       CALL T0
       LD DE,COPYR
       LD HL,23200    ; Line 21, column 0
       CALL T0        ; 3 lines just run on
       LD DE,64600                             ; GRUN
       LD HL,23138                             ; Line 18, column 2
       CALL G0                                 ; Display tall tree
       LD DE,64400                             ; GFIR
       LD HL,23141                             ; Line 19, column 5
       CALL G0                                 ; Display short tree
       LD DE,64400                            ; GFIR
       LD HL,23144                            ; Line 19, column 8
       CALL G0
       LD DE,64100                            ; GMOOR
       LD HL,23147                            ; Line 19, column11
       CALL G0                                ; Display moor symbol
       LD DE,63900                            ; GLAKE
       LD HL,23182                            ; Line 20, column 14
       CALL G0
       LD DE,63900                            ; GLAKE
       LD HL,23185                            ; Line 20, column 17
       CALL G0
       LD DE,64200                            ; GTOWN
       LD HL,23156                            ; Line 19, column20
       CALL G0
       LD DE,64400                            ; GFIR
       LD HL,23159                            ; Line 19, column 23
       CALL G0
       LD DE,64000                            ; GFLAG
       LD HL,23162                            ; Line 19, column 26
       CALL G0
       LD DE,64600                            ; GRUN
       LD HL,23165                            ; Line 19, column 29
       CALL G0
       LD HL,CONST
       LD DE,VARS
       LD BC,NCONS
       LDIR                ; Set first NCONS variables to values in CONST
       RET
VARS:  DEFS 64
CONST: DEFB 64             ; Initial values: bearing 90 degrees = 64b
       DEFB 0              ; sin b = 0
       DEFB 128            ; cos b = 1  (times 128)
       DEFB 0              ; Observer's start x = 1366.0
       DEFW 1366
       DEFB 0              ; Observer's start y = 2332.0
       DEFW 2332
       DEFS 12             ; 12 bytes do not need initialising
       DEFW 1438           ; x of finish
       DEFW 2272           ; y of finish
       DEFB 0              ; Clear the finished flag
NCONS: EQU 26              ; 26 bytes to move for initialisation
       END
```

names 1
2-5

Terrain & feature names

The section of memory from 63600 to 63737 inclusive contains text strings for 'textprint' to display (ASCII characters followed by Ø). These must be stored in the order given, which follows their numbering as terrain symbols & point feature names, ~~also~~ indexed elsewhere in the program.

The terrain types are

Ø	Run
1	Thick
2	Moor
3	Grass
4	Lake
5	Town

The point features are

Ø	Depression
1	Knoll
2	Niche
3	Boulder
4	Building
5	Ruin
6	Root␣stock
7	Sheep␣fold
8	Mine␣shaft
9	Rock␣outcrop
10	Pond
11	Water␣tank

vars 1
2-9

Layout of global variables (explanation - not coded).

```
            var    equ   bear
F7C7 63431  bear   defb      ; Bearing, ϕ..255
     63432  sinb   defb      ; 128*sin(bearing)
         3  cosb   defb      ; 128*cos(bearing)
         4  fxt    defb      ; fractional   }
         5  lxt    defb      ; low integer  } bytes of observer's x-coordinate
         6  hxt    defb      ; high integer }
         7  fyt    defb      ;   }
         8  lyt    defb      ;   } similar - observer's y-coordinate
         9  hyt    defb      ;   }
F7D0
     63440  fxh    defb      ;   }
         1  lxh    defb      ;   } similar - x-coordinate of terrain point to be evaluated
         2  hxh    defb      ;   }
         3  fyh    defb      ;   }
         4  lyh    defb      ;   } similar - y-coordinate     "     "
         5  hyh    defb      ;   }
         6  fhh    defb      ;   }
         7  lhh    defb      ;   } similar - result from height routine
         8  hhh    defb      ;   }
         9  fhg    defb      ;   }
     63450  lhg    defb      ;   } similar - height at observer
         1  hhg    defb      ;   }
         2  xf     defw      ; x-coordinate of finish (integer)
         4  yf     defw      ; y-  "    "
         6  finfl  defb      ; Flag : 1 means we have reached the finish
         7  colum  defb      ; Column no. for 'displa'
         8  tg     defb      ; Terrain symbol at observer } 2n : n = ϕ..5
         9  tery   defb      ; Terrain symbol on horizon  }
     63460  dbear  defb      ; Bearing to horizon point, ϕ..255.
         1  sind   defb      ; 128*sin (dbear)
         2  cosd   defb      ; 128*cos (dbear)
         3  obsfe  defb      ; Index to probability array for feature at observer.
```

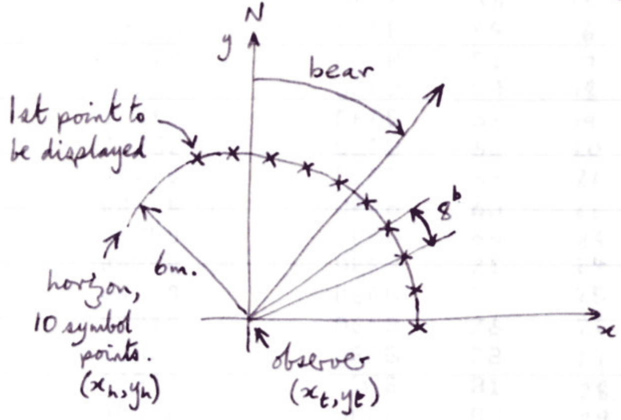
continued above right.

```
63464  obsco  defb   } Text string for flag
    5         defb   } code. 3rd byte always
    6         defb   } ϕ. 1st byte 32 ⇒ no
                       flag.
    7  spare  defs 3ϕ
```

Further notes on meaning of variables & their use.

Bearings are measured in byte-sized degrees, range ∅..255. This gives adequate resolution for orienteers:

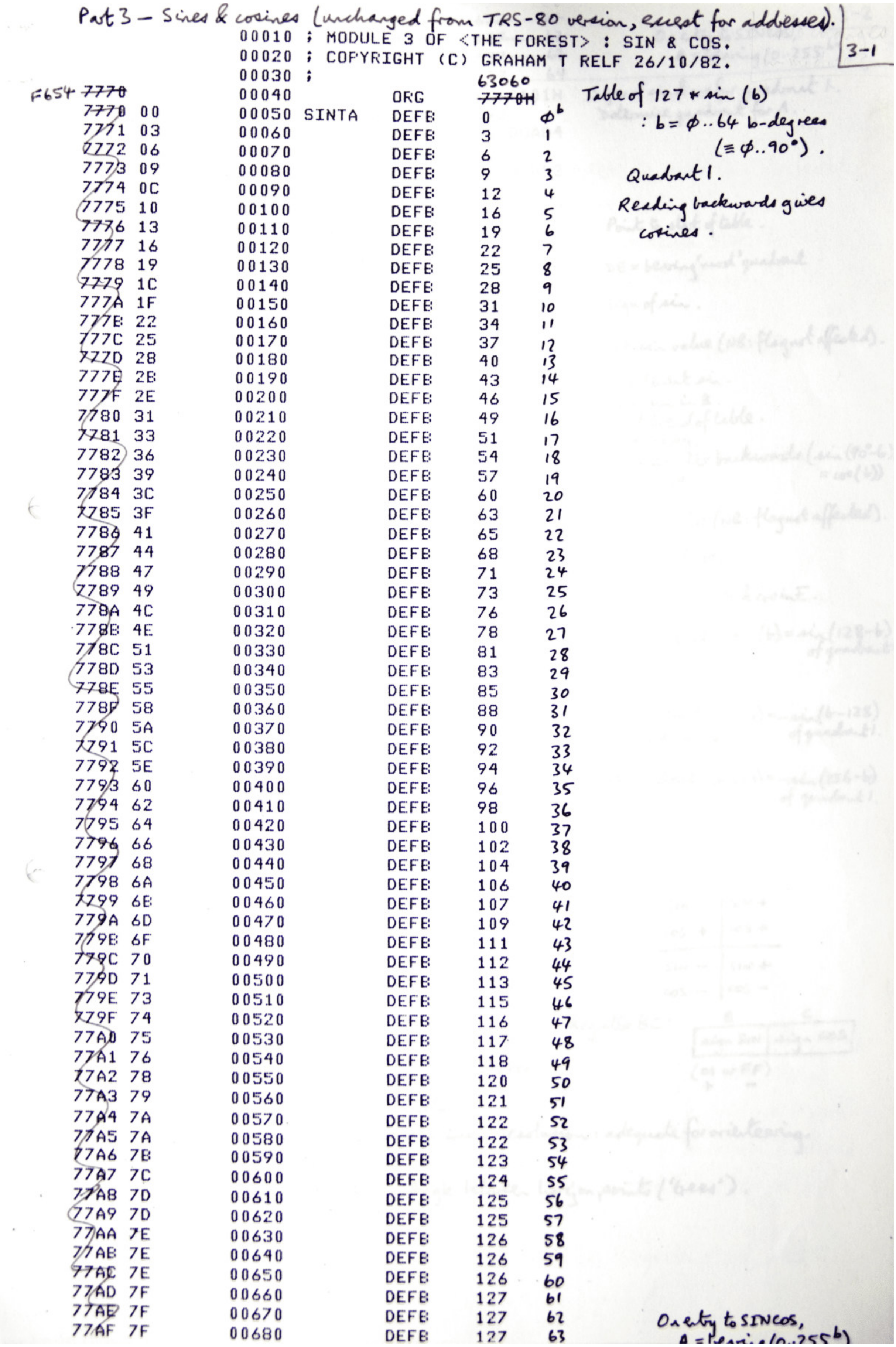
$$256^b \equiv 360°$$
$$64^b \equiv 90°$$
$$1^b \equiv 1°\!\cdot\!40625$$
$$8^b \equiv 11°\!\cdot\!25$$ is the spacing between graphic symbols in a displayed scene:


N
y
bear
1st point to be displayed
8^b
6m.
horizon, 10 symbol points. (x_h, y_h)
observer (x_t, y_t)
x

Coordinates of observer and horizon points, and heights are stored as 3-byte fixed-point numbers, always in metres:

HIGH	LOW .	FRAC

Arithmetic routines for such numbers are included in the program. Sines & cosines are looked up from a table, 'sinta', in which they are stored as signed integers in single bytes, as if multiplied by 128.



Part 3 — Sines & cosines (unchanged from TRS-80 version, except for addresses). 3-1

```
                 00010 ; MODULE 3 OF <THE FOREST> : SIN & COS.
                 00020 ; COPYRIGHT (C) GRAHAM T RELF 26/10/82.
                 00030 ;
F654 ~~7770~~    00040              ORG    63060 ~~7770H~~
7770  00         00050 SINTA        DEFB   0      ∅ᵇ
7771  03         00060              DEFB   3      1
7772  06         00070              DEFB   6      2
7773  09         00080              DEFB   9      3
7774  0C         00090              DEFB   12     4
7775  10         00100              DEFB   16     5
7776  13         00110              DEFB   19     6
7777  16         00120              DEFB   22     7
7778  19         00130              DEFB   25     8
7779  1C         00140              DEFB   28     9
777A  1F         00150              DEFB   31     10
777B  22         00160              DEFB   34     11
777C  25         00170              DEFB   37     12
777D  28         00180              DEFB   40     13
777E  2B         00190              DEFB   43     14
777F  2E         00200              DEFB   46     15
7780  31         00210              DEFB   49     16
7781  33         00220              DEFB   51     17
7782  36         00230              DEFB   54     18
7783  39         00240              DEFB   57     19
7784  3C         00250              DEFB   60     20
7785  3F         00260              DEFB   63     21
7786  41         00270              DEFB   65     22
7787  44         00280              DEFB   68     23
7788  47         00290              DEFB   71     24
7789  49         00300              DEFB   73     25
778A  4C         00310              DEFB   76     26
778B  4E         00320              DEFB   78     27
778C  51         00330              DEFB   81     28
778D  53         00340              DEFB   83     29
778E  55         00350              DEFB   85     30
778F  58         00360              DEFB   88     31
7790  5A         00370              DEFB   90     32
7791  5C         00380              DEFB   92     33
7792  5E         00390              DEFB   94     34
7793  60         00400              DEFB   96     35
7794  62         00410              DEFB   98     36
7795  64         00420              DEFB   100    37
7796  66         00430              DEFB   102    38
7797  68         00440              DEFB   104    39
7798  6A         00450              DEFB   106    40
7799  6B         00460              DEFB   107    41
779A  6D         00470              DEFB   109    42
779B  6F         00480              DEFB   111    43
779C  70         00490              DEFB   112    44
779D  71         00500              DEFB   113    45
779E  73         00510              DEFB   115    46
779F  74         00520              DEFB   116    47
77A0  75         00530              DEFB   117    48
77A1  76         00540              DEFB   118    49
77A2  78         00550              DEFB   120    50
77A3  79         00560              DEFB   121    51
77A4  7A         00570              DEFB   122    52
77A5  7A         00580              DEFB   122    53
77A6  7B         00590              DEFB   123    54
77A7  7C         00600              DEFB   124    55
77A8  7D         00610              DEFB   125    56
77A9  7D         00620              DEFB   125    57
77AA  7E         00630              DEFB   126    58
77AB  7E         00640              DEFB   126    59
77AC  7E         00650              DEFB   126    60
77AD  7F         00660              DEFB   127    61
77AE  7F         00670              DEFB   127    62
77AF  7F         00680              DEFB   127    63
```

Table of 127 * sin (b)
: b = ∅..64 b-degrees
(≡ ∅..90°).

Quadrant 1.

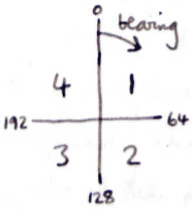
Reading backwards gives cosines.

On entry to SINCOS,
A = bearing (0..255ᵇ)

On entry to SINCOS, A = bearing ($0..255^b$).

```
77AD 7F        00660 (previous  ~~DEFB    127~~   ~~61~~
77AE 7F        00670   page)    ~~DEFB    127~~   ~~62~~
77AF 7F        00680            ~~DEFB    127~~   ~~63~~
-77B0 7F       00690 COSTA      DEFB      127     64
F695 ~~77B1~~ 010101    00700 SINCO   LD    BC,0101H    Signs of sin & cos for quadrant 1.
77B4 FEC0      00710            CP        192           Determine quadrant for A.
77B6 3038      00720            JR        NC,QUAD4
77B8 FE80      00730            CP        128
77BA 302D      00740            JR        NC,QUAD3
77BC FE40      00750            CP        64
77BE 3021      00760            JR        NC,QUAD2
77C0 217077    00770 QUAD1      LD        HL,SINTA      Point to start of table.
77C3 1600      00780            LD        D,0
77C5 5F        00790            LD        E,A           DE = bearing 'mod' quadrant
77C6 19        00800            ADD       HL,DE
77C7 78        00810            LD        A,B           Sign of sin.
77C8 FEFF      00820            CP        -1
77CA 7E        00830            LD        A,(HL)        Get sin value (NB: flag not affected).
77CB 2002      00840            JR        NZ,POSSI
77CD ED44      00850            NEG                     Complement sin.
77CF 47        00860 POSSIN     LD        B,A           Save sin in B.
77D0 21B077    00870            LD        HL,COSTA      Point to end of table.
77D3 A7        00880            AND       A             Clear carry.
77D4 ED52      00890            SBC       HL,DE         Offset pointer backwards (sin(90°-b)
77D6 79        00900            LD        A,C           Sign of cos.                = cos(b))
77D7 FEFF      00910            CP        -1
77D9 7E        00920            LD        A,(HL)        Get cos value (NB: flag not affected).
77DA 2002      00930            JR        NZ,POSCO
77DC ED44      00940            NEG                     Complement cos.
77DE 5F        00950 POSCO      LD        E,A
77DF 50        00960            LD        D,B           Return sin in D & cos in E.
77E0 C9        00970            RET
77E1 ED44      00980 QUAD2      NEG                     2nd quadrant : sin(b) = sin(128-b)
77E3 C680      00990            ADD       A,128                               of quadrant
77E5 0EFF      01000            LD        C,-1          cos < φ.
77E7 18D7      01010            JR        QUAD1
77E9 D680      01020 QUAD3      SUB       128           3rd quadrant: sin(b) = -sin(b-128)
77EB 01FFFF    01030            LD        BC,0FFFFH     sin < φ & cos < φ.      of quadrant 1.
77EE 18D0      01040            JR        QUAD1
77F0 ED44      01050 QUAD4      NEG                     4th quadrant: sin(b) = -sin(256-b)
77F2 06FF      01060            LD        B,-1          sin < φ.                of quadrant 1.
77F4 18CA      01070            JR        QUAD1
0000           01080            END
00000 TOTAL ERRORS
POSCOS   77DE
POSSIN   77CF
QUAD1    77C0
QUAD2    77E1
QUAD3    77E9
QUAD4    77F0
SINCOS   77B1
COSTAB   77B0
SINTAB   7770
```

Quadrants :



Signs:

SIN − COS +	SIN + COS +
SIN − COS −	SIN + COS −

Registers BC:

B	C
sign SIN	sign COS

(01 or FF)
 + −

All bearings are stored in 1 byte, as b-degrees:

$256^b = 360°$ Circle

$1^b = 1°.40625$ Minimum resolution: adequate for orienteering.

$64^b = 90°$

$8^b = 11°.25$ Angle between horizon points ('trees').

Part 4 – Height, terrain & feature generation (normal forest)

The height & other terrain attributes at any (x,y) point in the forest are calculated from functions, not stored as data. For an introduction to the philosophy behind this approach refer to my articles in 'Practical Computing', 5(3), March 82, pp. 93–95 and 5(9), Sept. 82, pp. 127–130.

The functions used in the 'normal' (simpler, fully mapped) forest are of the form

$$H = \sum_{i=1}^{5} \left[128 - \mathrm{abs}\left((a_i x + b_i y) \bmod 256 - 128 \right) \right]$$

in which x, y are coordinates of the point being evaluated (3-byte fixed-point numbers),

a_i, b_i are single-byte integer coefficients, 5 of each forming a table of 10 bytes, the table being different for each attribute being evaluated (height being counted as one attribute).

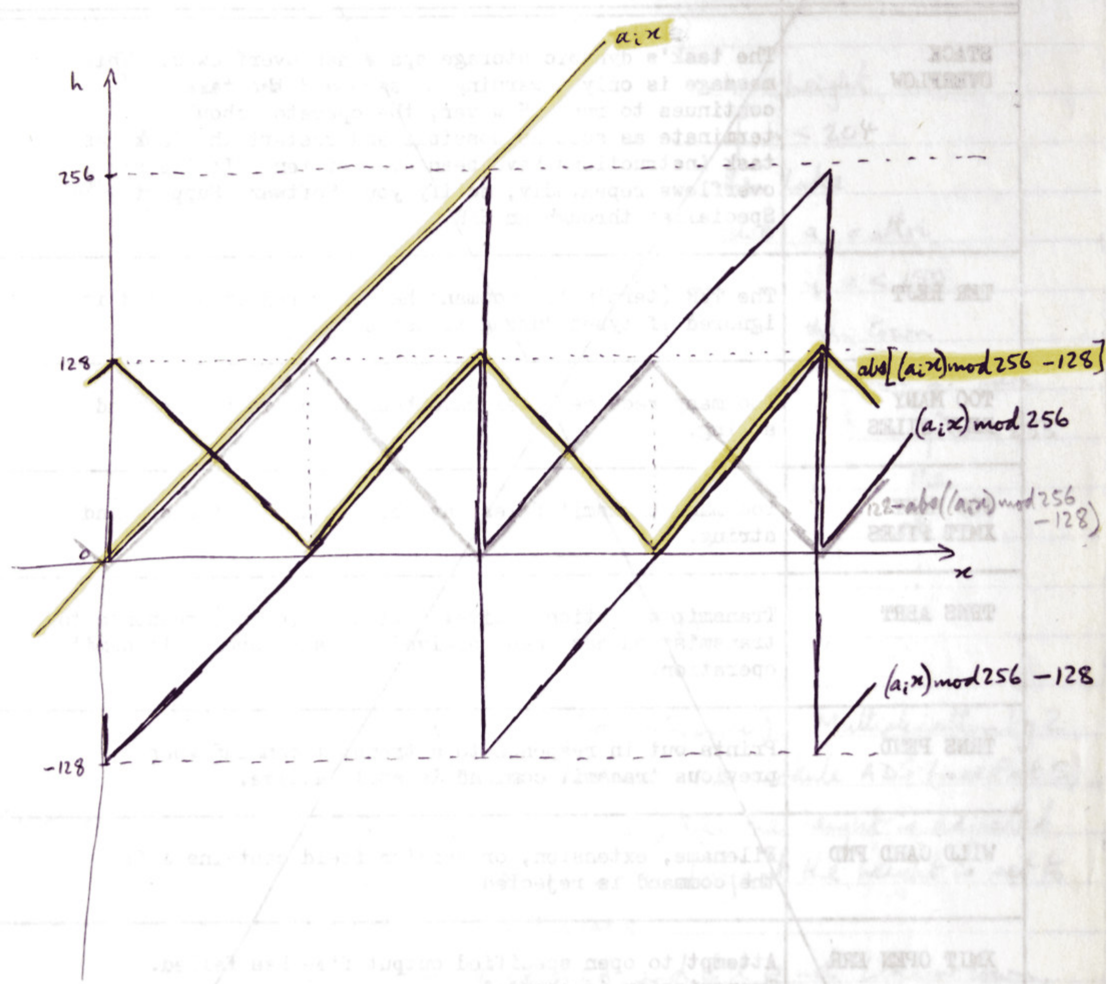
The functions are evaluated by a machine code routine whose main entry point is 'heigh' and subsequent re-entries are at 'attri'. Register IX always points to the table of coefficients a_i & b_i and must not be altered between re-entries to 'attri'.

The operations involving 256 and 128 in the formula above are acheived by permitting overflow in multiplications and by doing complements, for speed. These operations are the most processor-dependent parts of the program.

The diagram on the next page indicates how a saw-tooth function is generated for one value of i in the above formula. As a function of both x & y this would result in a linear wave pattern normal to the line $y = a_i x + b_i$. By summing 5 such patterns oriented in different directions a suitably random effect is achieved.

The more complex forest is generated more elegantly – see part 8.

Height / terrain attribute function for The Forest



$$H = \sum_{i=1}^{5} \left[128 - \text{abs}\left((a_i x + b_i y) \bmod 256 - 128\right)\right]$$

Initial values used in development:

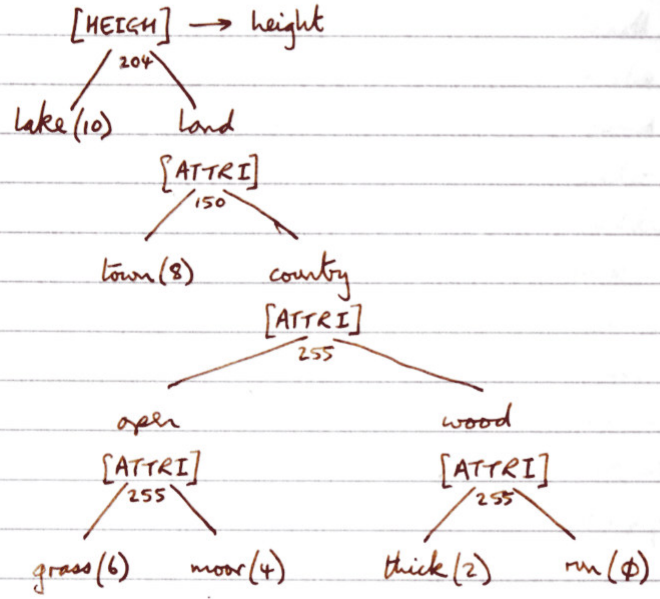
i	1	2	3	4	5
a_i	41	33	11	3	-13
b_i	2	7	29	37	27

(-13 stored as 243)

General form:

$$\mathcal{H} = \sum_i \text{abs}\left(\left(\sum_j a_{ij} x_j\right) \bmod K - \frac{K}{2}\right)$$

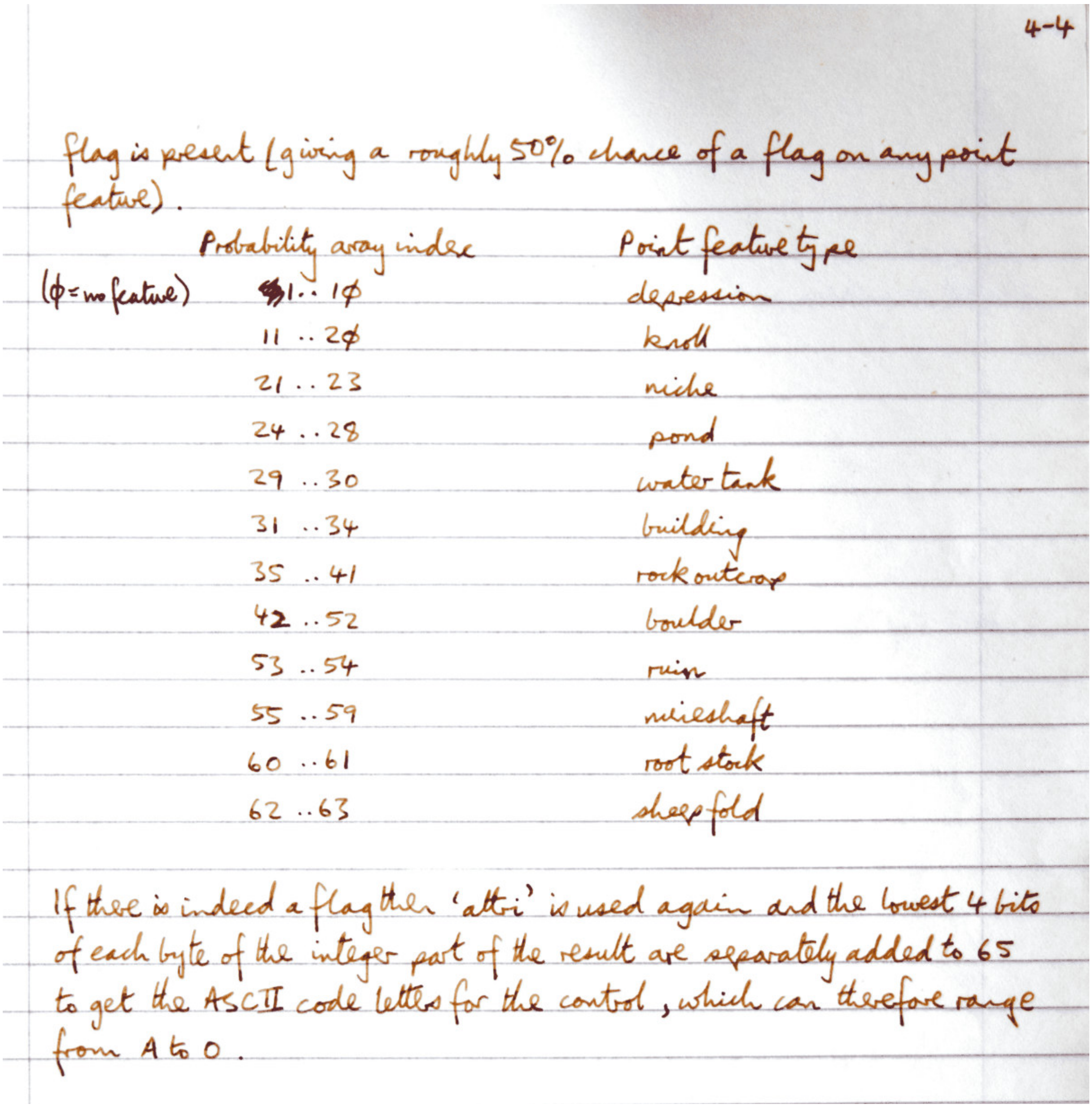
To determine the height and terrain type at any given point (x,y) the function may be evaluated up to 4 times, according to the following scheme.



```
h := height
if h < 204
then lake
else  a := altri
      if a < 150
      then town
           else a := altri
                if a < 255
                then
                etc.
                 :
```

Numbers in brackets are returned as 2×(terrain type no.). Multiplication by 2 is to index a word-based table of addresses in the module ADS (see Part 5). Note that if a lake is detected (height < 204) then the height is adjusted so we cannot go below the surface of the lake. In fact the height is set to 200 to give a small bank around the lake.

When the terrain type has been established, and if it is not lake or town, then the routine 'featu' is used to determine whether there is a point feature visible. To do this the fractional parts and the lowest 3 bits of the integer parts of the coordinates (x,y) of the point being tested are cleared to ∅. This ensures that point features are visible within 8×8 squares. The 'altri' routine is used yet again, this time with IX pointing half-way through one of the previously used tables of coefficients (a_i, b_i). If the result this time has bit 7 set but not bits ∅,1,2 or 4 then a point feature is present. If this is the case 'altri' is used again, with IX continuing from its last position. Bits ∅ to 5 index an array of probabilities of the different types of feature, as in the following table, while bit 6 determines whether a control



4-4

flag is present (giving a roughly 50% chance of a flag on any point feature).

	Probability array index	Point feature type
(ϕ = no feature)	1 .. 1ϕ	depression
	11 .. 2ϕ	knoll
	21 .. 23	niche
	24 .. 28	pond
	29 .. 30	water tank
	31 .. 34	building
	35 .. 41	rock outcrop
	42 .. 52	boulder
	53 .. 54	ruin
	55 .. 59	mineshaft
	60 .. 61	root stock
	62 .. 63	sheepfold

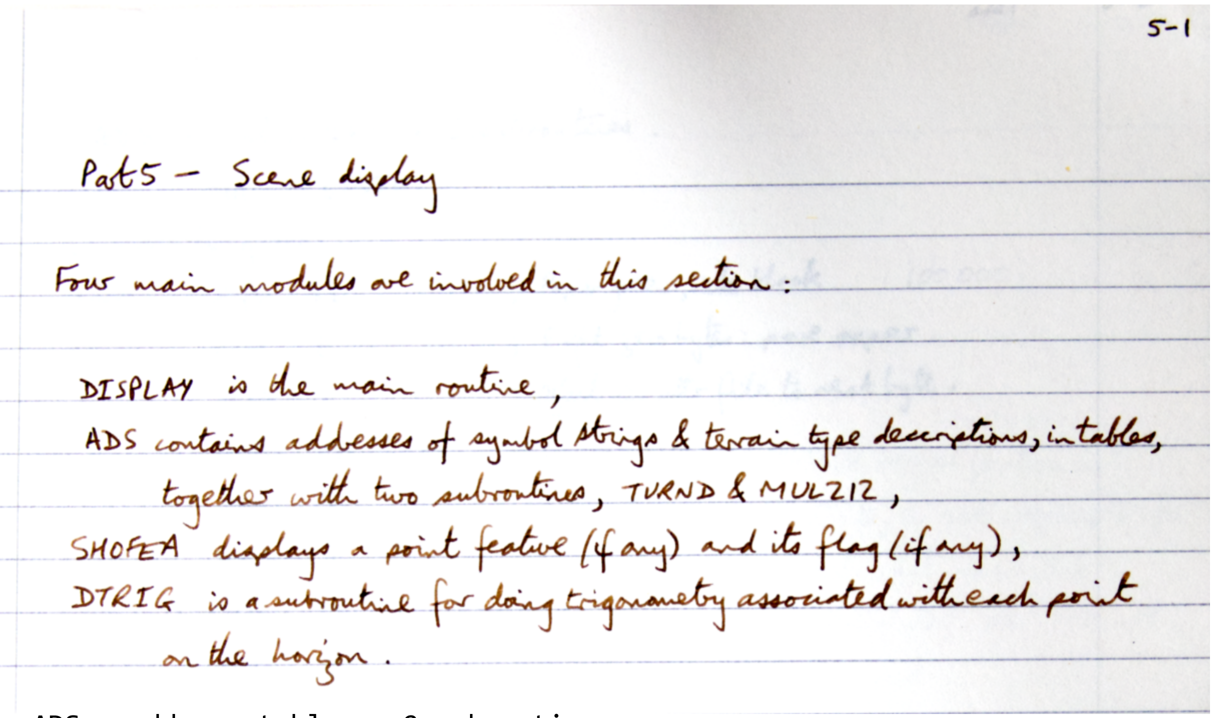
If there is indeed a flag then 'altri' is used again and the lowest 4 bits of each byte of the integer part of the result are separately added to 65 to get the ASCII code letters for the control, which can therefore range from A to O.

See later for details of how the terrain generator is modified for the improved “Complex Forest”.

```
       ORG 62580     ;TERRA - the original terrain generator
FXH:   EQU 63440
LXH:   EQU FXH + 1
HXH:   EQU FXH + 2
FYH:   EQU FXH + 3
LYH:   EQU FXH + 4
HYH:   EQU FXH + 5
FHH:   EQU FXH + 6
LHH:   EQU FXH + 7
HHH:   EQU FXH + 8
TG:    EQU FXH + 18
TERSY: EQU FXH + 19
TERRA: CALL HEIGH
       LD A,(FH)
       LD (FHH),A
       LD HL,(LH)
       LD (LHH),HL
       LD A,H
       CP 0
       JR NZ,LAND
       LD A,L
       CP 204        ;Lake height + bank
       JR C,TLAKE    ;Lake here
LAND:  XOR A         ;A = 0
       LD (FXH),A    ;Attributes to constant
       LD (FYH),A    ;over 1m cells
       CALL ATTRI    ;NB: HEIGH left IX pointing to TABA1
       LD A,(HH)
       CP 0
       JR NZ,CNTRY
       LD A,(LH)
       CP 150
       JR C, TTOWN   ;Town here
CNTRY: CALL ATTRI    ;NB: Last ATTRI left IX pointing to TABA2
       LD A,(HH)
       CP 0
       JR NZ,WOOD
       LD A,(LH)
       CP 255
       JR C,OPEN
WOOD:  CALL ATTRI    ;NB: Last ATTRI left IX pointing to TABA3
       LD A,(HH)
       CP 0
       JR NZ,TRUN    ;Runnable wood here
       LD A,(LH)
       CP 255
       JR C,THICK    ;Thicket here
TRUN:  LD A,0
       LD (TERSY),A
       JP FEATU
THICK: LD A,2
       LD (TERSY),A
       JP FEATU
```

```
OPEN:  CALL ATTRI    ;NB: Last ATTRI left IX pointing to TABA3
       LD A,(HH)
       CP 0
       JR NZ,TMOOR   ;Moor here
       LD A,(LH)
       CP 255
       JR C,TGRAS
TMOOR: LD A,4
       LD (TERSY),A
       JP FEATU
TGRAS: LD A,6
       LD (TERSY),A
       JP FEATU
TTOWN: LD A,8
       LD (TERSY),A
       JP FEATU
TLAKE: LD A,200      ;Set lake level so we cannot
       LD (LHH),A
       XOR A
       LD (FHH),A
       LD A,10
       LD (TERSY),A
       JP FEATU
;Height & attributes function:
HEIGH: LD IX,ABH     ;Point to table of height coefficients
ATTRI: XOR A         ;A = 0
       LD (FH),A     ;Initialise
       LD (LH),A     ;the
       LD (HH),A     ;sum.
       LD B,5        ;No of coeff pairs in table
HLOO1: PUSH BC       ;Save loop counter i
       LD DE,(LXH)   ;Get
       LD A,(FXH)    ;x-coordinate
       LD B,A
       CALL MULCO    ;HL.A = a[i]*x, point IX to b[i]
       PUSH HL
       PUSH AF
       LD DE,(LYH)   ;Get
       LD A,(FYH)    ;y-coordinate
       LD B,A
       CALL MULCO    ;HL.A = b[i] * y, point IX to a[i+1]
       POP BC
       POP DE        ;DE.B = a[i] * x
       ADD A,B
       ADC HL,DE     ;HL.A = a[i]x + b[i]y
NORM1: LD D,L
       LD E,A        ;D.E = result mod 256
       LD A,D        ;MSB
       CP 127        ;If in right half of triangle
       JR C,REPOS
       LD A,E        ;then fold over (ie, negate D.E)
       CPL
       ADD A,1
       LD E,A
       LD A,D
       CPL
       ADC A,0
       LD D,A        ;D.E = 256 - D.E
REPOS: LD HL,(FH)    ;Add into overall sum in FH,LH,HH
       ADD HL,DE
       LD (FH),HL
       LD A,(HH)
       ADC A,0
       LD (HH),A
       POP BC        ;Retrieve loop counter, i
       DJNZ HLOO1    ;Next i
       RET
;Multiply coord DE.B by signed coeff pointed by IX
;Result in HL.A
MULCO: XOR A         ;A = 0
       LD H,A
       LD L, A
       LD C,(IX+0)   ;Get coeff a[i] or b[i]
       BIT 7,C       ;Sign?
       JR NZ,SUBCO
       CALL ADDBT
       CALL ADDBT
       CALL ADDBT
       CALL ADDBT
       CALL ADDBT
       CALL ADDBT
       CALL ADDBT    ;7 times
ENDCO: RR H
       RR L
       RRA
       INC IX        ;Point to next coeff, b[i] or a[i+1]
       RET
SUBCO: CALL SUBBT
       CALL SUBBT
       CALL SUBBT
       CALL SUBBT
       CALL SUBBT
       CALL SUBBT
       CALL SUBBT    ;7 times
       JR ENDCO
ADDBT: RR H
       RR L
       RRA
       RR C
       RET NC
       ADD A,B
       ADC HL,DE
       RET
SUBBT: RR H
       RR L
       RRA
       RR C
       RET NC
       SUB B
       SBC HL,DE
       RET
;Point features, flags & codes
FEATY: DEFB 0        ;Index to 1..63 array of probabilities of
;each feature type (0..11)
CODE:  DEFB 32       ;2-letter flag code,
       DEFB 32       ;spaces if no flag
       DEFB 0
FEATU: LD A,(TERSY)  ;Get observer's terrain type
       CP 7
       JR NC,NOFEA   ;No features in lake or town
       LD IX,TABA1-5 ;Re-use existing coeff tables
       XOR A         ;A = 0
       LD (FXH),A    ;Features to be visible in 8x8m area:
       LD (FYH),A
       LD A,(LXH)
       AND 0F8H
       LD (LXH),A
       LD A,(LYH)
       AND 0F8H
       LD (LYH),A
       CALL ATTRI    ;Results in FH,LH,HH
       LD A,(LH)     ;NB: FH = 0 so can't use that
       AND 097H      ;10010111
       CP 128
       JR NZ,NOFEA
       CALL ATTRI    ;NB: Last ATTRI left IX at TABA2-5
       LD A,(LH)
       AND 63        ;Indexing 1..63 array
       LD (FEATY),A
       LD A,(LH)
       AND 64
       JR Z,NOCOD    ;1 in 2 chance of a flag
       CALL ATTRI    ;NB: Last ATTRI left IX at TABA3-5
       LD A,(HH)
       AND 15        ;Range A..O [in practice A..C]
       ADD A,65      ;To ASCII letter
       LD (CODE),A
       LD A,(LH)
       AND 15        ;Range A..O
       ADD A,65
       LD (CODE+1),A
       RET
NOFEA: XOR A         ;A = 0
       LD (FEATY),A
NOCOD: LD A,32
       LD (CODE),A
       LD (CODE+1),A
       RET
;Local data:
```

```
FH:     DEFB 0
LH:     DEFB 0
HH:     DEFB 0
;Tables of coefficients:
ABH:    DEFB 200        ;a1
        DEFB 3          ;b1
        DEFB 165        ;a2
        DEFB 37         ;b2
        DEFB 127        ;a3
        DEFB 96         ;b3
        DEFB 101        ;a4
        DEFB 202        ;b4
        DEFB 13         ;a5
        DEFB 157        ;b5
TABA1:  DEFB 171
        DEFB 131
        DEFB 171
        DEFB 140
        DEFB 184
        DEFB 106
        DEFB 97
        DEFB 126
        DEFB 124
        DEFB 96
TABA2:  DEFB 152
        DEFB 107
        DEFB 153
        DEFB 182
        DEFB 68
        DEFB 162
        DEFB 118
        DEFB 179
        DEFB 158
        DEFB 171
TABA3:  DEFB 179
        DEFB 172
        DEFB 190
        DEFB 108
        DEFB 186
        DEFB 101
        DEFB 192
        DEFB 192
        DEFB 95
TABAD:  DEFB 172
        END
```



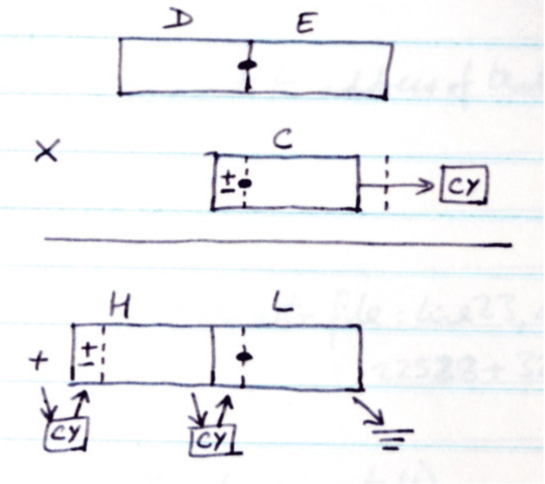
5-1

Part 5 – Scene display

Four main modules are involved in this section:

DISPLAY is the main routine,
ADS contains addresses of symbol strings & terrain type descriptions, in tables, together with two subroutines, TURND & MULZ12,
SHOFEA displays a point feature (if any) and its flag (if any),
DTRIG is a subroutine for doing trigonometry associated with each point on the horizon.

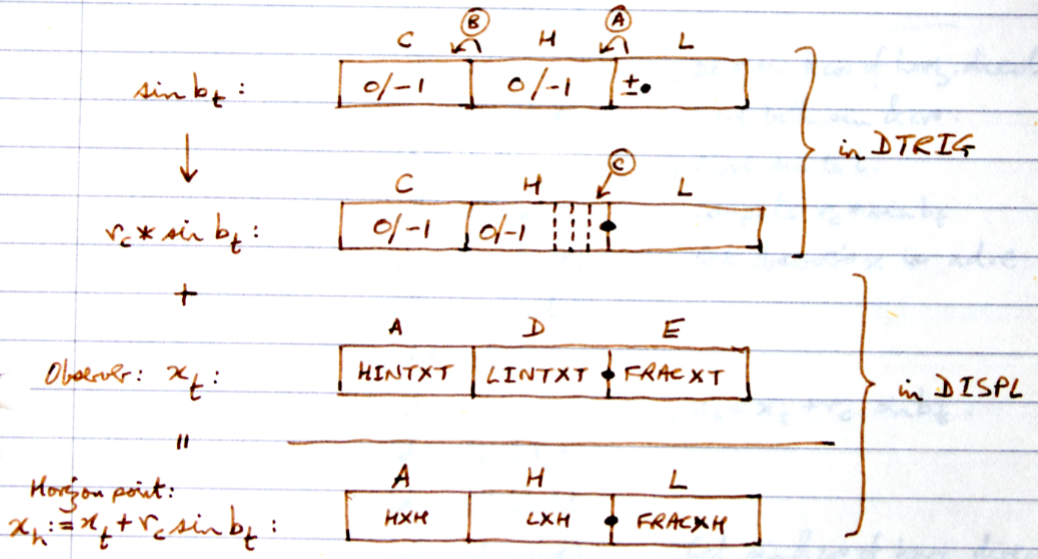
```
; ADS - address tables + 2 subroutines
        ORG 62350
; Graphic for grass:
GRASS:  DEFB 32     ; Paper green, ink black
        DEFS 8      ; 8 zero bytes - pure paper
        DEFB 1      ; Offset in attr file to next byte
        DEFB 32
        DEFS 8
        DEFB 1
        DEFB 32
        DEFS 8
        DEFB 0
; The next 2  tables are indexed by 2t, where t = terrain type, 0..5
ADSYM:  DEFW 64600  ; Table of graphics
        DEFW 64400
        DEFW 64100
        DEFW GRASS
        DEFW 64200
        DEFW 63900
ADOBS:  DEFW 63600  ; Table of terrain messages
        DEFW 63604
        DEFW 63610
        DEFW 63615
        DEFW 63626
        DEFW 63621

BEAR:   EQU 63431
SINB:   EQU 63432
SINCO:  EQU 63125
TURND:  LD A,(BEAR) ; Compute & store sin & cos of new bearing
        CALL SINCO  ; after a turn
        LD HL,SINB
        LD (HL),D
        INC HL
        LD (HL),E
        RET
; Multiply +ve 2-byte fixed-point no in D.E by 1-byte signed fraction in C
; [+/- .7 bits], to get signed 2-byte fixed-point in H.L [+/- 7Bits.byte].
; NB: D.E not changed
MUL212: LD A,1      ; Sign +ve until otherwise
        BIT 7,C     ; If sign(C) < 0
        JR Z,POS212
        LD A,C      ; then
        NEG         ; complement and
        LD C,A      ; set -ve sign
        LD A,255    ; for result
POS212: LD B,7      ; 7 bits left in C to shift & add
        LD HL,0
        AND A       ; Clear carry
LO212:  RR H
        RR L
        OR C
        JR NC,EN212 ; If bit set at right of C
        ADD HL,DE   ; then add DE into result. NB: Carry affected
EN212:  DJNZ LO212  ; NB: Loop test preserves carry
        RR H        ; Shift to put any
        RR L        ; carry into result
        CP 255      ; If sign (in A) +ve
        RET NZ      ; then done
NEG2:   LD A,L      ; else take
        CPL         ; twos complement
        ADD A,1     ; of result
        LD L,A
        LD A,H
        CPL
        CPL
        ADC A,0
        LD H,A
        AND 80H     ; Select bit 7 (sign)
        RLCA
        NEG
        LD B,A      ; HL >= 0 => B = 0 else B = -1
        RET
        END
; NEG2 may be used as general subroutine
; to complement HL
```



```
; SHOFEA - show point feature & flag
       ORG 62495
T0:    EQU 63757
SHOFE: LD HL,FIN1
       LD B,12          ; 12 feature types, counter
SHLOO: CP (HL)          ; If feature index in A > table entry
       JR C,INC2        ; then this is the feature type
       INC HL
       INC HL           ; Word-based table => 2 increments
       DJNZ SHLOO
       RET
INC2:  LD DE,FIN2-FIN1
       ADD HL,DE        ; HL points to address of text string
       LD E,(HL)
       INC HL
       LD D,(HL)
       LD HL,23283      ; Point to attr file, line 23, column 19:
       CALL T0          ; 22528 + 32 * 23 + 19
       RET
; Index1: probability table for point features (see part 4)
FIN1:  DEFW 11
       DEFW 21
       DEFW 24
       DEFW 29
       DEFW 31
       DEFW 35
       DEFW 42
       DEFW 53
       DEFW 55
       DEFW 60
       DEFW 62
       DEFW 64
; Index 2: string addresses
FIN2:  DEFW 63631       ; "Depression"
       DEFW 63642       ; "Knoll"
       DEFW 63648       ; "Niche"
       DEFW 63722       ; "Pond"
       DEFW 63727       ; "Water tank"
       DEFW 63662       ; "Building"
       DEFW 63709       ; "Rock outcrop"
       DEFW 63654       ; "Boulder"
       DEFW 63671       ; "Ruin"
       DEFW 63698       ; "Mine shaft"
       DEFW 63676       ; "Root stock"
       DEFW 63687       ; "Sheep fold"
       END
```

DTRIG — Subroutine to compute $r_c * \begin{Bmatrix} \sin \\ \cos \end{Bmatrix}$(horizon point direction). ↖ (in separate calls).

```
          org   62480
dtrig     ld    h,0
          bit   7,l         ; (A)
          jr    z,dpos
          ld    h,255
dpos      ld    c,h         ; (B)
          ld    b,4         ; r_c = 8 = 2^(4-1)
dloo2     add   hl,hl       ; (C) HL := r_c * sin b_t
          djnz  dloo2
          ret
          end
```



Similarly for $\cos b_t$, y_t, y_h in DISPL.

```
       ORG 62050     ; DISPL - display current scene
CHLIN: DEFB 0
SCLIN: DEFB 0
TURN:  CALL TURND
DISPL: LD A,12       ; Paper blue, ink green
       LD (ATTR0),A
       CALL CLS      ; Clear screen
       LD A,2        ; 1st col to display (tree can overhang col 1)
       LD (COLUM),A
       LD A,(BEAR)   ; Get straight-ahead bearing
       SUB 36        ; Look 4.5 symbol spaces left
       LD (DBEAR),A
DLOO1: LD A,(DBEAR)
       CALL SINCO    ; DE = sin & cos of horizon direction
       LD (SIND),DE  ; Save both sin & cos
       LD L,D        ; Move sin to L
       CALL DTRIG    ; Compute rc * sin bt
       LD DE,(FXT)   ; Get observer's x in AD.E
       LD A,(HXT)
       ADD HL,DE
       ADC A,C       ; xh = xt + rc * sin bt
       LD (HXH),A    ; Horizon position
       LD (FXH),HL
       LD HL,(SIND)  ; Get sin & cos of horizon direction
       CALL DTRIG    ; Compute rc * cos bt
       LD DE,(FYT)   ; Get observer's y in AD.E
       LD A,(HYT)
       ADD HL,DE
       ADC A,C
       LD (HYH),A
       LD (FYH),HL
       CALL TERRA    ; Get height, terrain, feature for horizon point
       LD A,(FHH)
       LD C,A
       LD A,(FHG)
       SUB C
       LD HL,(LHG)
       LD DE,(LHH)
       SBC HL,DE     ; HL.A = (hg-hh)*sh = ht difference, sh = 1 => range
-23..24
       LD BC,24
       ADD HL,BC     ; HL.A += 24, range 0..47
       LD H,L
       LD L,A        ; H.L = ht diff, range 47..0
       LD A,H
       SRL A         ; A = ht diff, range 23..0
       LD (CHLIN),A  ; Character-sized position
       XOR A         ; A = 0
       LD (HFLAG),A  ; No flag until shown otherwise
       LD A,(CODE)
       CP 32         ; Is flag visible from horizon point?
       JR Z, NFLAG
       LD A,(OBSCO)
       CP 32          ; Has observer got a flag too?
       JR NZ,NFLAG
       LD A,1
       LD (HFLAG),A   ; Horizon flag to be shown
NFLAG: LD A,(TERSY)   ; Get terrain type (x 2)
       LD C,A
       LD B,0         ; BC = word offset from ADSYM, graphics table
       LD HL,ADSYM
       ADD HL,BC
       LD E,(HL)
       INC HL
       LD D,(HL)      ; DE points to graphic
       LD A,(CHLIN)   ; A = row, 0..23
       LD H,0
       LD L,A         ; HL = row, 0..23
       ADD HL,HL      ; x 32:
       ADD HL,HL
       ADD HL,HL
       ADD HL,HL
       ADD HL,HL      ; HL = 32 * row
       LD A,(COLUM)
       LD B,0
       LD C,A         ; BC = column
       ADD HL,BC      ; HL = 32 * row + column
       LD BC,22528    ; Start of attributes file
       ADD HL,BC      ; HL points to row & col in attr file
       LD A,(TERSY)   ; Terrain symbol
       CP 6           ; Grass => no graphic
       JR Z,NG0
       PUSH HL
       CALL G0        ; Display graphic
       POP HL
       LD A,(TERSY)
       CP 10          ; Lake => no flag on horizon nor ground bar
       JR Z,NGRAF
NG0:   LD DE,31
       ADD HL,DE      ; HL points to start of ground (grass) in attr file
       LD DE,62350    ; DE points to grass in ADS
       CALL G0        ; Display grass
       LD A,(HFLAG)
       CP 0           ; No horizon flag => no more on this horizon point
       JR NZ,NGRAF
       LD DE,34
       AND A
       SBC HL,DE      ; HL points to horizon flag position
       LD A,H
       CP 91          ; If off screen don't show it
       JR NC,NGRAF
       LD (HL),3BH    ; Paper white, ink magenta
NGRAF: LD A,(COLUM)   ; Get column
       CP 28          ; If > 28, all done
       JR NC,ENDIS
       ADD A,3        ; Else next column
       LD (COLUM),A
       LD A,(DBEAR)
       ADD A,8       ; Next horizon point bearing
       LD (DBEAR),A
       JP DLOO1      ; Next horizon point (loop)
ENDIS: LD HL,22895   ; Line 11, column 15 = centre of screen
       SET 7,(HL)    ; Flash = level marker
       LD A,(TG)     ; Get observer's terrain type (x 2)
       LD HL,ADOBS   ; Offset pointer from ADOBS, message table
       LD C,A
       LD D,0
       ADD HL,DE
       LD E,(HL)
       INC HL
       LD D,(HL)     ; DE points to message string
       LD HL,23264   ; Display at line 23 column 0
       CALL T0       ; Display observer's terrain message
       LD A,(OBSFE)
       CP 0          ; If no feature at observer, then done
       JR Z,ENDAL
       CALL SHOFE    ; Show feature
       LD HL,(OBSCO)
       LD A,32
       CP H          ; If no flag at observer, then done
       JR Z,ENDAL
       LD DE,64000   ; Point to graphic for flag
       LD HL,23279   ; Line 23 column 15
       CALL G0       ; Display flag
ENDAL: RET
HFLAG: DEFB 0        ; Whether flag is visible from horizon point
       END



       ORG 62040  ; HIGH - returns int (height) for test purposes
HEIGH: EQU 62737
LH:    EQU 63011
HIGH:  CALL HEIGH
       LD BC,(LH)
       RET
       END
```

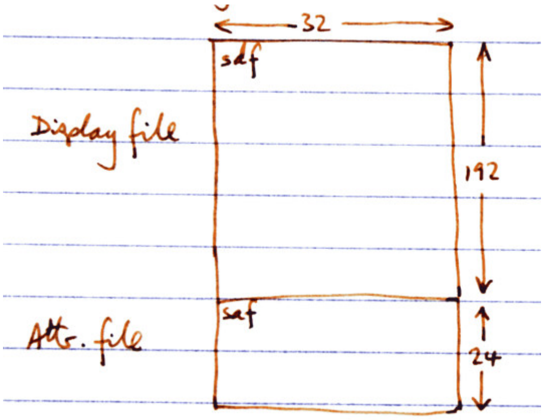
Part 6 - Turning and moving forward

Two modules:

SCROLLS - move screen left and right in character sized steps (not pixel sized because colours would go funny; colours are in the attributes file which is character based).

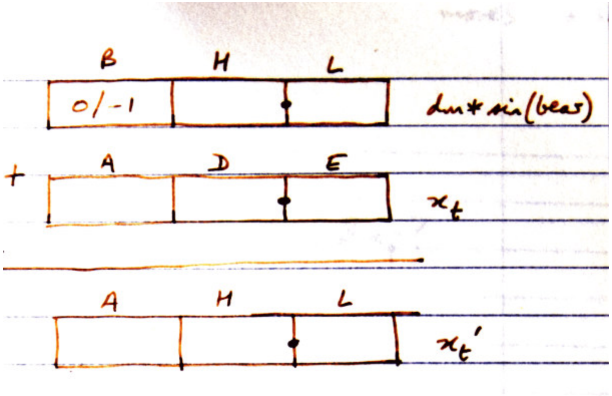
MOVE - moves the observer forward one pace (about 1m +/- 10%) and drifts the bearing by up to +/- 1 b-degree.

In scrolling we have to move both the display file and the attributes file. The display files is moved first, to minimise the period when the colours might be wrong.



```
       ORG 61700       ; SCROLLS - left & right scrolling (panning)
TURN:  EQU 62052
ATTR0: EQU 63850
BEAR:  EQU 63431
RIGHT: LD A,(BEAR)     ; Turn right 8 b-degrees
       ADD A,8
       LD (BEAR),A
       LD B,3          ; Scroll left 3 times = 1 symbol space
LEFLO: PUSH BC
       CALL SCLEF
       POP BC
       DJNZ LEFLO
       JP TURN
LEFT:  LD A,(BEAR)     ; Turn left 8 b-degrees
       SUB 8
       LD (BEAR),A
       LD B,3          ; Scroll right 3 times
RIGLO: PUSH BC
       CALL SCRIG
       POP BC
       DJNZ RIGLO
       JP TURN
BACK:  LD A,(BEAR)     ; Half turn, 128 b-degrees
       ADD A,128
       LD (BEAR),A
       LD B,30         ; Scroll left 30 times
       JR LEFLO
; Start addresses of display & attr files:
SDF:   EQU 16384
SAF:   EQU 22528
; Scroll left by 1 character:
SCLEF: LD HL,SDF
       CALL CLCOL      ; Clear left column
       LD BC,6143      ; Bytes to move
       LD DE,SDF       ; Destination
       LD HL,SDF+1     ; Source
       LDIR            ; Move display file
       LD BC,767
       LD DE,SAF
       LD HL,SAF+1
       LDIR            ; Move attr file
       RET
; Scroll right by 1 character:
SCRIG: LD HL,SDF+31
       CALL CLCOL      ; Clear right column
       LD BC,6143
       LD DE,SDF+6143
       LD HL,SDF+6142
       LDDR            ; Move display file
       LD BC,767
       LD DE,SAF+767
       LD HL,SDF+766
       LDDR            ; Move attr file
       RET
; Clear column whose top is pointed to by HL:
CLCOL: LD B,192
       LD DE,32
CL1:   LD (HL),0       ; Zero display file (paper everywhere!)
       ADD HL,DE
       DJNZ CL1
       LD A,(ATTR0)    ; (Might depend on whether colour TV?)
       LD B,24
CL2:   LD (HL),A       ; Reset attributes in attr file
       ADD HL,DE
       DJNZ CL2
       RET
       END
```

```
        ORG 61850     ; MOVE - move observer forward and drift
BEAR:   EQU 63431
TURND:  EQU 62404
MUL212: EQU 62417
TERRA:  EQU 62580
SINB:   EQU 63432
COSB:   EQU 63433
FXT:    EQU 63434
LXT:    EQU 63435
HXT:    EQU 63436
FYT:    EQU 63437
LYT:    EQU 63438
HYT:    EQU 63439
FXH:    EQU 63440
LXH:    EQU 63441
HXH:    EQU 63442
FYH:    EQU 63443
LYH:    EQU 63444
HYH:    EQU 63445
FHH:    EQU 63446
LHH:    EQU 63447
HHH:    EQU 63448
FHG:    EQU 63449
LHG:    EQU 63450
HHG:    EQU 63451
XF:     EQU 63452
YF:     EQU 63454
FINFL:  EQU 63456
TG:     EQU 63458
TERSY:  EQU 63459
FEATY:  EQU 62905
OBSFE:  EQU 63463
CODE:   EQU 62906
OBSCO:  EQU 63464
DISPL:  EQU 62055
MOVE:   LD A,R        ; Get random byte from Z80 refresh register
        LD C,A        ; treat as fraction to be multiplied by 0.1
        LD D,0
        PUSH DE       ; Save fraction while bearing drifts
        LD A,(BEAR)
        BIT 5,C
        JR Z,NPLUS
        ADD A,1       ; Inc bearing by 1 b-degree
NPLUS:  BIT 3,C
        JR Z,NOMIN
        SUB 1         ; Dec bearing by 1 b-degree
NOMIN:  LD (BEAR),A
        CALL TURND    ; Revise sin & cos
        POP DE        ; Retrieve random fraction
        LD C,1AH      ; C = 0.1
        CALL MUL212   ; HL = random part of distance to move
        LD DE,0E6H    ; DE = fixed part of same = 0.9
        ADD HL,DE     ; HL = distance to move
        PUSH HL         ; Save for incrementing yt
        EX DE,HL        ; Move to DE for incrementing xt
        LD A,(SINB)     ; sin(BEAR)
        LD C,A
        CALL MUL212     ; HL = sinb * distance to move
        LD DE,(FXT)     ; Adding to xt : get fractional & low parts of xt
        LD A,(HXT)      ; Get high part of xt
        ADD HL,DE
        ADC A,B         ; Use sign byte from LU"!" result above
        LD (FXT),HL     ; xt = xt + move * sinb
        LD (FXH),HL     ; New xt will be used to find obeserver's height
        LD (HXT),A
        LD (HXH),A
```

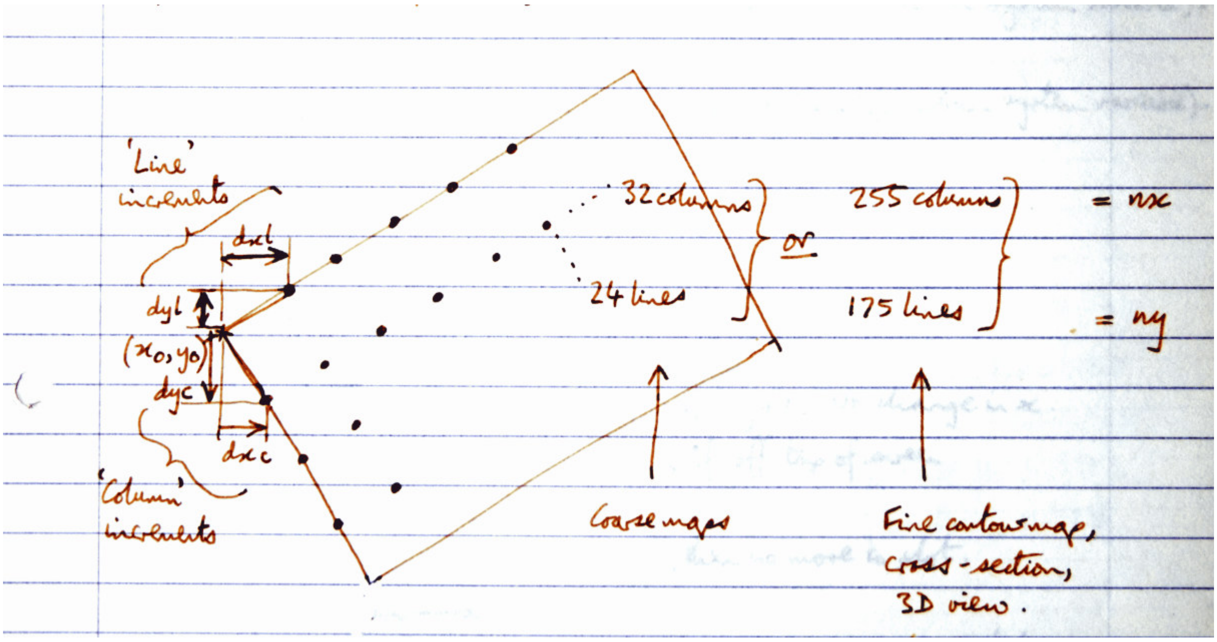


```
; Similarly for yt:
        POP DE
        LD A,(COSB)
        LD C,A
        CALL MUL212
        LD DE,(FYT)
        LD A,(HYT)
        ADD HL,DE
        ADC A,B
        LD (FYT),HL
        LD (FYH),HL
        LD (HYT),A
        LD (HYH),A
        LD HL,YF+1      ; Test for finish:
        CP (HL)         ; Do MSBs of yt & yf match?
        JR NZ,HMOVE
        LD A,(LYT)
        AND 0FCH
        LD B,A
        DEC HL
        LD A,(HL)
        AND 0FCH
        CP B            ; And LSBs, within 4 metres?
        JR NZ,HMOVE
        DEC HL
        LD A,(HXT)
        CP (HL)         ; And MSBs of xt and xf?
        JR NZ,HMOVE
        LD A,(LXT)
        AND 0FCH
        LD B,A
        DEC HL
        LD A,(HL)
        AND 0FCH
        CP B
        JR NZ,HMOVE     ; And LSBs within 4 metres?
        LD A,1          ; If all matched we are at finish
        LD (FINFL),A    ; Set flag for BASIC shell to test
HMOVE:  CALL TERRA      ; Get observer's height and terrain
        LD A,(FHH)      ; Save height in HG
        LD (FHG),A
        LD HL,(LHH)
        LD (LHG),HL
        LD A,(TERSY)  ; Save terrain type in TG
        LD (TG),A
        LD A,(FEATY)  ; Save feature type
        LD (OBSFE),A
        LD HL,(CODE)  ; Save flag code
        LD (OBSCO),HL
        JP DISPL
```

Part 7 - Maps

The first routine in this section, DRAWFN, is a subroutine for plotting or unplotting points in a vertical line according to a binary pattern held in array BAR. This is used to draw contoured cross-sections and 3D views (which are essentially repeated cross-sections).

Maps are drawn by calling a standard harness (entry FHEAD) having first poked the address of a kernel routine into MADDR from the BASIC shell. Different types of map are produced by different kernels. The kernel determines the map content for a particular point (x, y), while the harness scans a rectangular area, the definition of which has been poked from the BASIC shell:



x_0, y_0 is top left, to draw the map in normal reading order, not bottom left as set up by the user in the BASIC shell.

A cross-section is actually done by using the VIEW3D kernel with ny = 1 (ie, just 1 line to draw).

```
       ORG 61500     ; DRAWFN subroutine
PLOT:  EQU 22E5H     ; Spectrum Rom routine. Enter with B = y, C = x
BAR:   DEFS 100      ; Standard contoured line, preset
X0:    DEFB 0        ; Screen position for
Y0:    DEFB 0        ; start of line
LEN:   DEFB 0        ; Length to plot
PBAR:  Ld HL,BAR
       LD A,(LEN)
       LD D,A        ; Counter
       LD BC,(X0)    ; BC = y0 * 256 + X0
PLOOP: LD A,(HL)
       CP 0
       JR Z,BLACK
       SET 2,(IY+87) ; Inverse video (Spectrum system variable)
       JR DO
BLACK: RES 2,(IY+87) ; Normal video (Spectrum system variable)
DO:    PUSH HL
       PUSH DE
       PUSH BC
       CALL PLOT
       POP BC
       POP DE
       POP HL
       INC B         ; Y = y + 1. No change in x
       LD A,175      ; If off top of screen
       CP B
       JR C,PEND     ; then no more to plot
       DEC D         ; Count point just plotted
       XOR A
       CP D          ; If LEN points done
       JR Z,PEND     ; then no more
       INC HL        ; Next point on bar
       JR PLOOP
PEND:  RET
       END


       ORG 61300     ; MAPS - harness, scans rectangular area to be mapped
X0:    DEFW 0        ; Start of plot, top left on map
Y0:    DEFW 0
XL:    DEFW 0        ; Start of current line on map
YL:    DEFW 0
XH:    EQU 63441     ; Current position on map (global coordinates)
YH:    EQU 63444
DXL:   DEFW 0        ; Map increments along scan line, as diagram above
DYL:   DEFW 0
DXC:   DEFW 0        ; MAP increments between scans (ie, column)
DYC:   DEFW 0
NX:    DEFB 0        ; No of points in scan line, 0..255 or 0..31
NY:    DEFB 0        ; No of lines, 0..191 or 0..23
XS:    DEFB 0        ; Current screen x, 0..255
YS:    DEFB 0        ; Current screen y, 191..0
XX:    DEFB 0        ; Current screen character x, 0..31
```

```
YX:    DEFB 0        ; Current screen character y, 0..23
MADDR: DEFW 0        ; Address of 1-point kernel for current map type -
                     ; poke from BASIC first
FHEAD: LD HL,(X0)    ; Start first line
       LD (XL),HL
       LD HL,(Y0)
       LD (YL),HL
       XOR A
       LD (XH-1),A   ; No fractional positions
       LD (YH-1),A
       LD A,(NY)     ; No of lines
       LD B,A        ; B = line counter
       LD A,191      ; Line start on screen (175 for FINEC)
       LD (YS),A
       XOR A
       LD (YX),A
FLINE: PUSH BC       ; Start any line
       LD HL,(XL)    ; Work point at line start
       LD (XH),HL
       LD HL,(YL)
       LD (YH),HL
       LD A,(NX)     ; No of points in line
       LD B,A        ; B = point counter
       XOR A
       LD (XS),A     ; Line start on screen
       LD (XX),A
FPT:   PUSH BC       ; Do one point
       LD HL,(MADDR) ; Get address of kernel for current map type
       JP (HL)       ; and go there
FRET:  POP BC        ; returning always to here
       DJNZ NEXTP
       POP BC
       DJNZ NEXTL
       RET
; Next point in a line:
NEXTP: LD HL,(XH)
       LD DE,(DXL)
       ADD HL,DE
       LD (XH),HL
       LD HL,(YH)
       LD DE,(DYL)
       ADD HL,DE
       LD (YH),HL
       LD HL,XX      ; Increment character no
       INC (HL)      ;          (to handle all map types)
       LD HL,XS      ; Increment pixel no
       INC (HL)
       JR FPT        ; Do next point
; Next line:
NEXTL: LD HL,(XL)
       LD DE,(DXC)
       ADD HL,DE
       LD (XL),HL
       LD HL,(YL)
       LD DE,(DYC)
       ADD HL,DE
       LD (YL),HL
       LD HL,YX
       INC (HL)      ; Character lines count up from 0 at top
       LD HL,YS
       DEC (HL)      ; Pixel lines count down from 191 at top
       JR FLINE      ; Do next line
       END
```

Now for some kernels…

```
       ORG 61240  ; VIEW3D - a map kernel - (MADDR) = VIEW
HEIGH: EQU 62737
LH:    EQU 63011
X0:    EQU 61600
Y0:    EQU 61601
LEN:   EQU 61602
PBAR:  EQU 61603
XS:    EQU 61318
YS:    EQU 61319
FRET:  EQU 61385
VIEW:  CALL HEIGH ; (LH) = int (height (XH, YH))
       LD HL,(LH)
       LD BC,204  ; Lake height
       AND A      ; Clear carry
       SBC HL,BC  ; HL = HL - lake height
       SRA H
       RR L
       SRA H
       RR L
       SRA H
       RR L       ; HL = HL / 8
       LD A,L
       CP 128     ; If < 0
       JR C,HPOS
       XOR A      ; then A = 0
HPOS:  INC A
       LD (LEN),A
       LD A,(YS)  ; 191..0
       SUB 70
       LD (Y0),A  ; (i0 in BASIC version)
       LD A,(XS)
       LD (X0),A
       CALL PBAR
       JP FRET    ; End of kernel
       END
```

The next one, FINEC, is very similar:

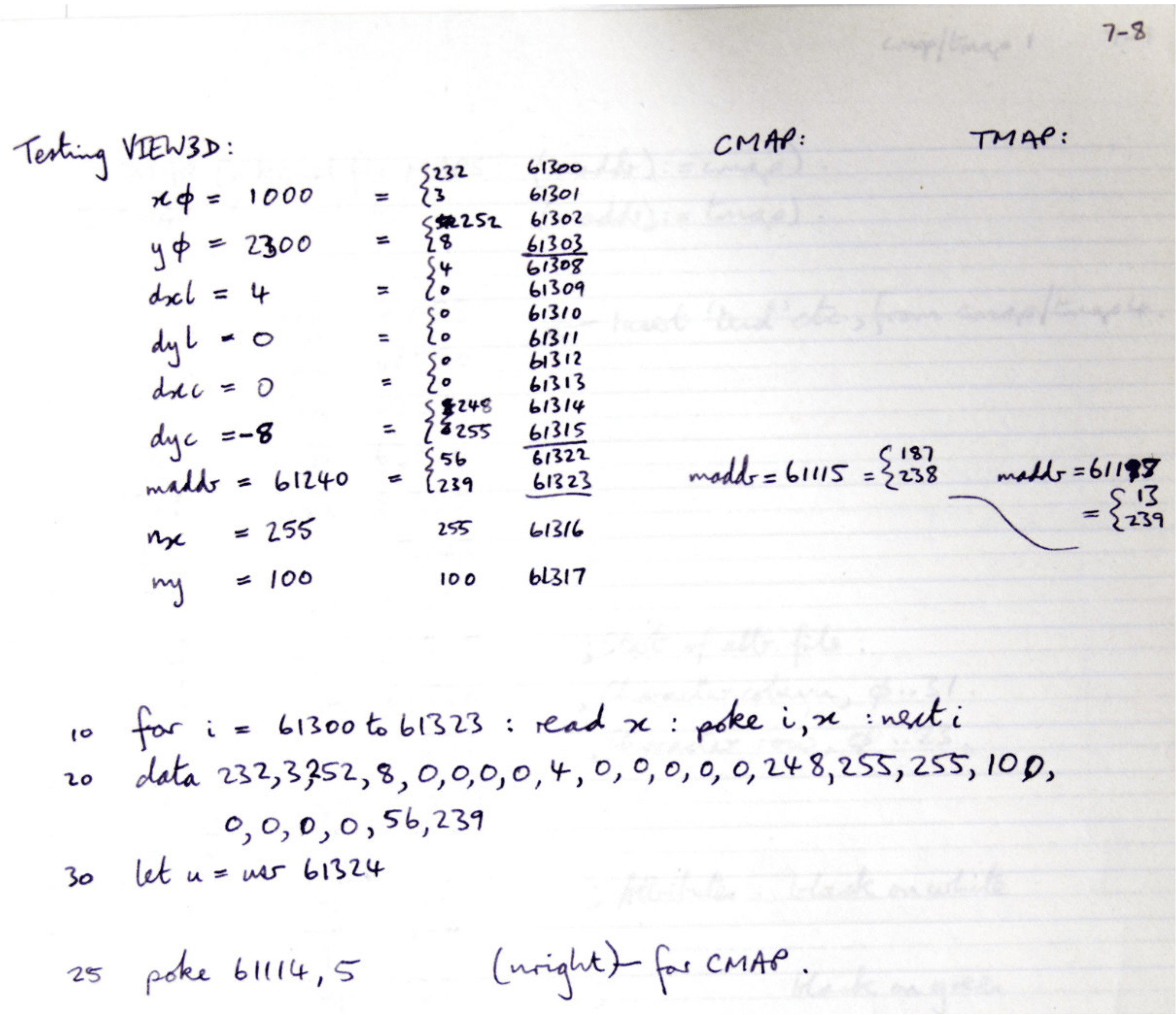
```
       ORG 61240  ; FINEC - fine contours kernel - (MADDR) = FINEC
HEIGH: EQU 62737
PLOT:  EQU 22E5H  ; (In ROM)
BAR:   EQU 61500
LH:    EQU 63011
X0:    EQU 61600
Y0:    EQU 61601
LEN:   EQU 61602
PBAR:  EQU 61603
XS:    EQU 61318
YS:    EQU 61319
FRET:  EQU 61385
VIEW:  CALL HEIGH ; (LH) = int (height (XH, YH))
       LD HL,(LH)
       LD BC,204  ; Lake height
       AND A      ; Clear carry
       SBC HL,BC  ; HL = HL - lake height
       SRA H
       RR L
       SRA H
       RR L
       SRA H
       RR L       ; HL = HL / 8
       LD A,L
       CP 128     ; If < 0
       JR C,HPOS
       XOR A      ; then A = 0
HPOS:  INC A
       LD H,0
       LD L,A     ; 191..0
       LD BC,BAR
       ADD HL,BC  ; HL indexes BAR array
       LD A,(HL)  ; Value to plot, from BAR
       CP 0
       JP Z,FRET  ; IF white, no plot
       LD BC,(XS) ; (XS) in C, (YS) in B
       CALL PLOT
       JP FRET    ; End of kernel
       END
```

To make fine contours map from BASIC:

```
8500 LET NX=255:LET NY=175:POKE 61322,72:POKE 61323,238:
      POKE 61348,175:GOTO 9500
```

and

```
6000 CLS:INPUT “Fine or coarse map? [f/c]”;F$:IF F$=”f” THEN GOTO 8500
6100 LET NX=32:LET NY=24:POKE 61322,187:POKE 61323,238:GOTO 9500
```



7-8

Testing VIEW3D:

```
xφ   = 1000     = {232    61300
                  {3      61301
yφ   = 2300     = {252    61302
                  {8      61303
dxl  = 4        = {4      61308
                  {0      61309
dyl  = 0        = {0      61310
                  {0      61311
dxc  = 0        = {0      61312
                  {0      61313
dyc  =-8        = {248    61314
                  {255    61315
madds = 61240   = {56     61322
                  {239    61323
nx   = 255        255     61316
ny   = 100        100     61317
```

CMAP:

madds = 61115 = {187
{238

TMAP:

madds = 61197
= {13
{239

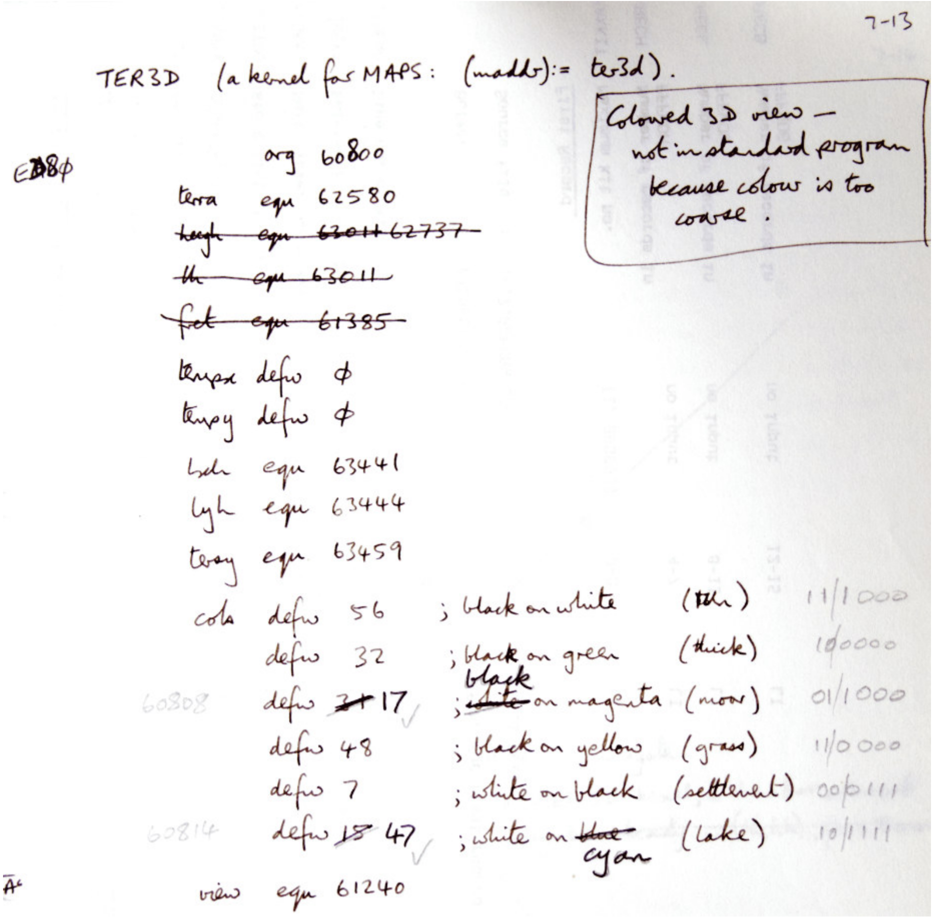
```
10  for i = 61300 to 61323 : read x : poke i,x :next i
20  data 232,3,252,8,0,0,0,0,4,0,0,0,0,0,248,255,255,100,
         0,0,0,0,56,239
30  let u = usr 61324

25  poke 61114,5          (nright) for CMAP.
```

The next chunk of code includes 2 kernels. CMAP is the coarse contour map and TMAP is the terrain map.

```
        ORG 61088       ; CMAP & TMAP kernels
HEIGH:  EQU 62737
LH:     EQU 63011
FRET:   EQU 61385
TERRA:  EQU 62580
TATTR:  EQU 63745
T0:     EQU 63757
SAF:    EQU 22528       ; Start of attr file
XX:     EQU 61320       ; Character column, 0..31
YX:     EQU 61321       ; Character row, 0..23
LXH:    EQU 63442       ; Global position on map
LYH:    EQU 63445
TERSY:  EQU 63459
SYMS:   DEFM "r"
        DEFB 56         ; Black on white
        DEFM "t"
        DEFB 32         ; Black on green
```

```
       DEFM "m"
       DEFB 31       ; White on magenta
       DEFM "g"
       DEFB 48       ; Black on yellow
       DEFM "s"
       DEFB 7        ; White on black
       DEFM "l"
       DEFB 15       ; White on blue
NRIGH: DEFB 5        ; No of right shifts to scale height (must not be 0)
CHAR:  DEFB 0
       DEFB 0        ; End of CHAR string, for T0
CMAP:  CALL HEIGH
       LD DE,(LH)
       LD A,D
       CP 0
       JR NZ,SHIFT
       LD A,E
       CP 204        ; Lake height
       JR NC,SHIFT
       LD E,200      ; If lake replace with surface (as in TERRA)
SHIFT: LD A,(NRIGH)
       LD B,A
CLOOP: SRL D
       RR E          ; E = height to plot
       DJNZ CLOOP
       LD A,'<'      ; Lowest height band "letter"
       ADD A,E
       AND 7FH       ; No characters above 128!
       LD (CHAR),A   ; To display using T0
       AND 7         ; Paper colour
       ADD A,A
       ADD A,A
       ADD A,A       ; Moves paper colour to bits 3, 4, 5
       CP 40         ; 101000 - check contrast for letter
       JR NC,SHOW    ; Ink black
       OR 7          ; Ink white
SHOW:  LD (TATTR),A
       LD A,(YX)
       LD L,A
       LD H,0
       ADD HL,HL
       ADD HL,HL
       ADD HL,HL
       ADD HL,HL
       ADD HL,HL     ; 5 times: HL = HL * 32
       LD A,(XX)
       LD C,A
       LD B,0
       ADD HL,BC
       LD BC,SAF
       ADD HL,BC     ; HL points to attr file position for T0
       LD DE,CHAR    ; Point to string to display
       CALL T0
```

```
        JP FRET       ; End of CMAP kernel
TMAP:   LD HL,(LXH)   ; Terrain map kernel
        LD (TEMPX),HL ; Save coordinates because FEATU alters them
        LD HL,(LYH)
        LD (TEMPY),HL
        CALL TERRA
        LD HL,(TEMPX)
        LD (LXH),HL
        LD HL,(TEMPY)
        LD (LYH),HL
        LD A,(TERSY)
        LD C,A
        LD B,0
        LD HL,SYMS
        ADD HL,BC
        JP TEND
; Rest of TMAP code went to start of module because out of space to next
; module
        ORG 61088
TEND:   LD A,(HL)     ; Character to display
        LD (CHAR),A
        INC HL
        LD A,(HL)     ; Attributes
        JR SHOW
TEMPX:  DEFW 0
TEMPY:  DEFW 0
        END
```



7-13

TER3D (a kernel for MAPS: (maddr):= ter3d).

Coloured 3D view — not in standard program because colour is too coarse.

```
EB8φ          org  60800
         terra   equ  62580
         ~~heigh  equ  63011 62737~~
         ~~th     equ  63011~~
         ~~feat   equ  61385~~
         tempx   defw  φ
         tempy   defw  φ
         lxh     equ  63441
         lyh     equ  63444
         tersy   equ  63459
         cols    defw  56        ; black on white       (thin)       11/1000
                 defw  32        ; black on green       (thick)      10/0000
60808            defw  ~~31~~ 17 ✓   ; ~~white~~ black on magenta  (moor)   01/1000
                 defw  48        ; black on yellow      (grass)      11/0000
                 defw  7         ; white on black       (settlement) 00/0111
60814            defw  ~~15~~ 47 ✓   ; white on ~~blue~~ cyan   (lake)      10/1111
Ā
         view    equ  61240
```

```
EDΦ0   ter3D:        ld   hl,(bch)
                     ld   (tempx),hl
                     ld   hl,(lyh)
                     ld   (tempy),hl
                     call terra
                     ld   hl,(tempx)
                     ld   (bch),hl
                     ld   hl,(tempy)
                     ld   (lyh),hl
                     ld   a,(teray)
  (                  ld   c,a
                     ld   b,Φ               ✓
         60849       ld   hl,~~[illegible]~~ cols
         60852       add  hl,bc          9
                     ~~add hl,bc~~
         60853       ld   a,(hl)         126
    EDB6=60854       ld   (23695),a          ; ATTR-T, in system.
                     jp   view               ; To VIEW3D kernel, to draw bar~~, except
                          ✓                    height is already in (th,hh), from terra~~.

EA9C
(  BC
```

```
          2
33   13~~3~~  237
↑     ↑
21   85   ED
```

~~50 + 256*143 =~~

```
60854   50                                    X
    5  143 ]                           141 ]
    6   92 ]— 23695                     92 ]— 23693  ≡ ATTR_P
              ≡ATTR_T
    7  195                                        10.37.45
    8   59                                     6: 10.35.35
    9  239                              56         -------
        ↓                              239            2.10
      61243                             ↓
        X                             61240       A:  10.43.43
                                        ✓             10.38.10
                                                      --------
                                                        5.33
```

Part 8 – The complex forest

The forest can be changed by altering the tables of coefficients which are used in terrain generation (see Part 4). This does not alter the angular nature of the terrain however, with its plane surfaces. The switch to the so-called complex forest involves changing the machine code ~~from~~ between addresses 62783 and 62802 inclusive, and the use of a table of 256 values stored between 60500 and 60755. The table forms a profile, shown on the next page, which replaces the simple saw-tooth function used in generating the normal forest (see p. 4-2). This means that the height/attributes generating routine no longer computes

$$h_i = 128 - \text{abs}((a_i x + b_i y) \bmod 256 - 128)$$

but instead looks up

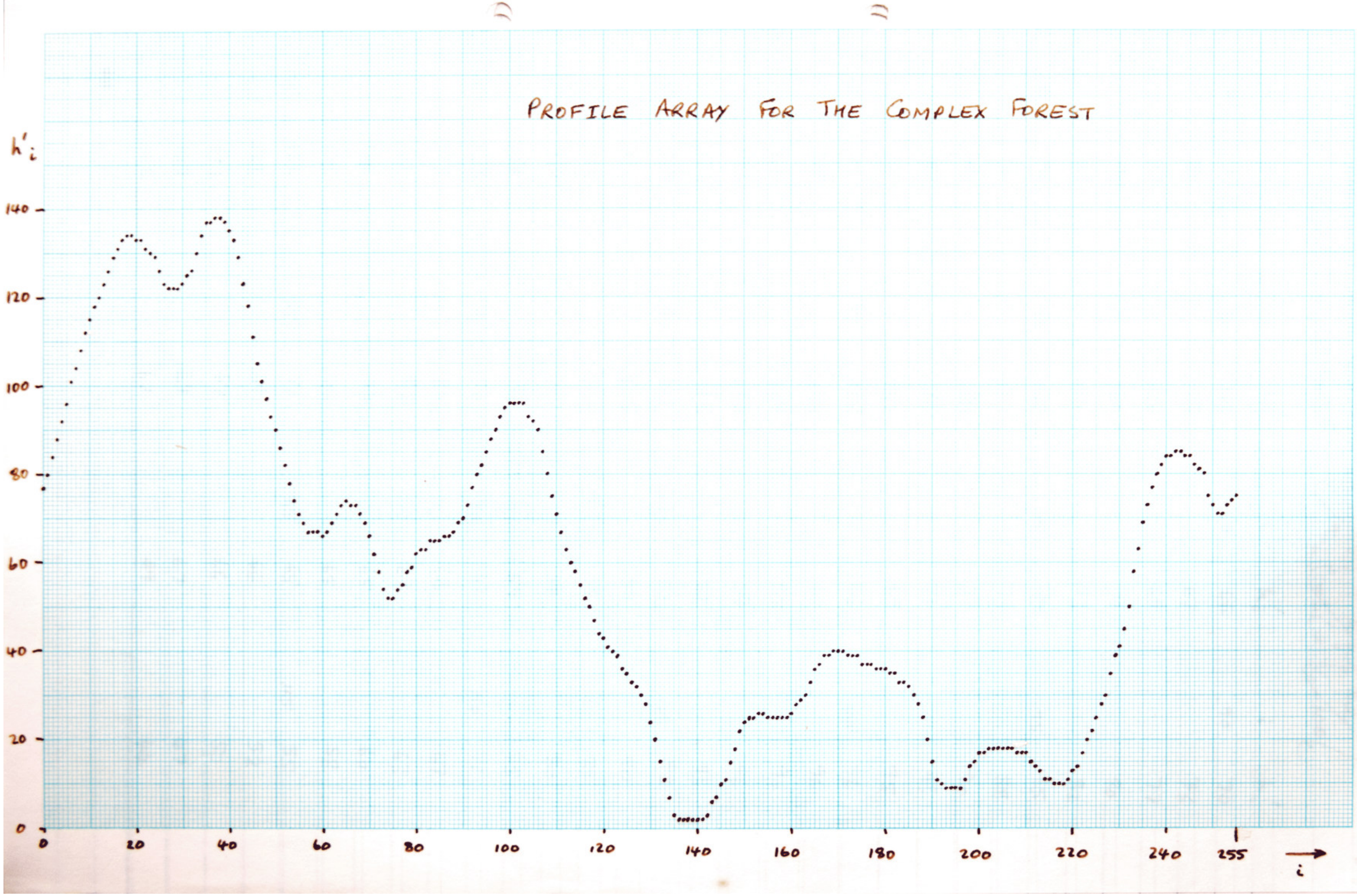
$$h'_i((a_i x + b_i y) \bmod 256)$$

in the profile table (array), to form $H' = \sum_{i=1}^{5} h'_i$.

The assembly program between labels norm1 and ~~norm2~~ rpos+2 (inclusive) on page 4-8 is therefore replaced by the following when the user invokes the complex forest. This code and the original are swapped by poke statements from the Basic shell – see page 9-9.

```
comp1   ld     h,Ø          ; L := result mod 256.
        ld     bc,fstbt     ; Point to start of profile array, at 60500.
        add    hl,bc
        ld     c,(hl)       ; BC := h'i .
        ld     b,Ø
        ld     hl,(hh)
        add    hl,bc        ; Add into sum for H'.
        ld     (hh),hl
        pop    bc           ; Retrieve loop counter, i (as normal).
        djnz   hloo1        ;Next i
```

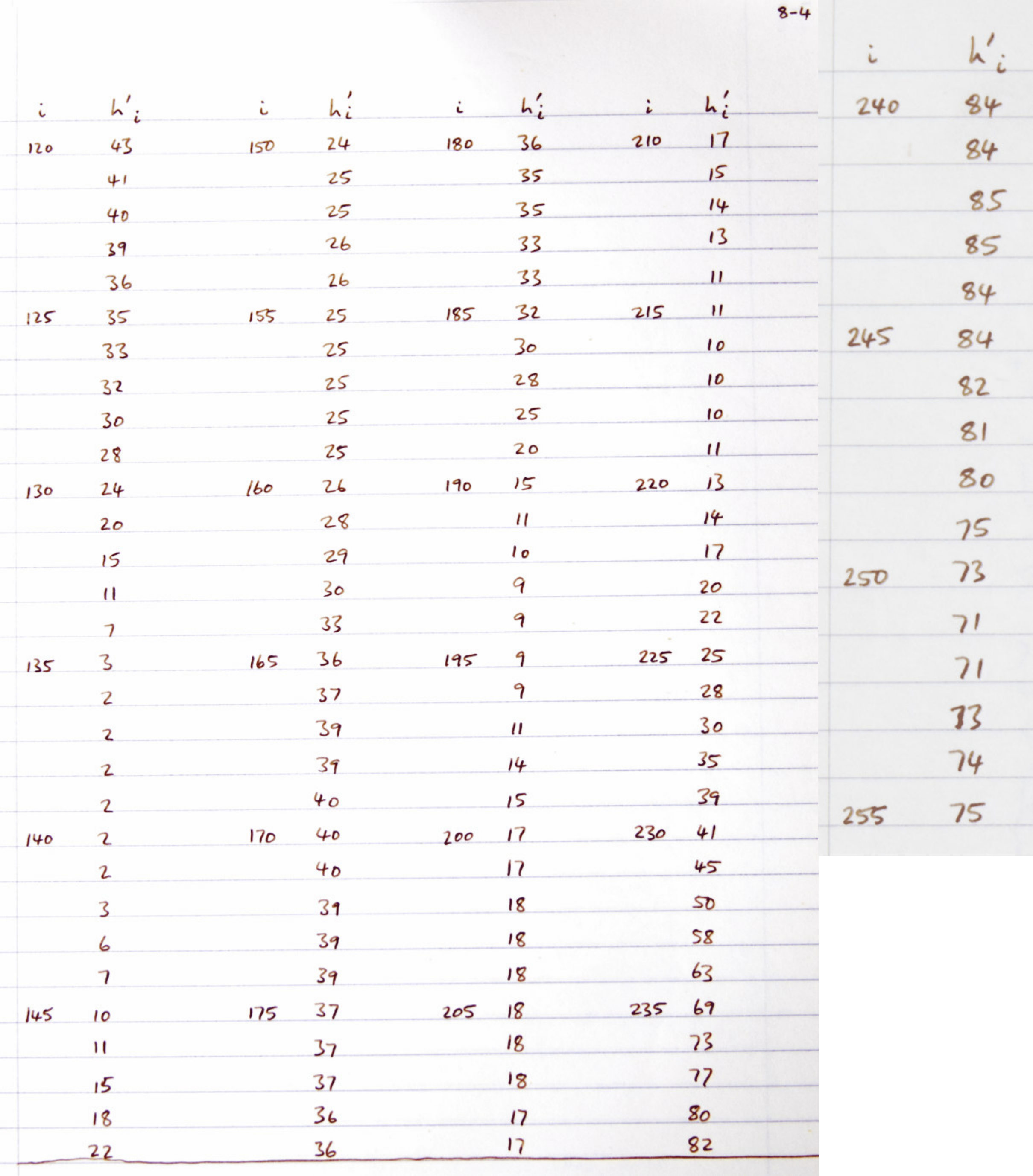
PROFILE ARRAY FOR THE COMPLEX FOREST



~~NB: These numbers should be scaled by . . . . ? to get the mapped Complex Forest.~~ ~~↖ $h_i := h_i' * 1.36 - 100;$~~

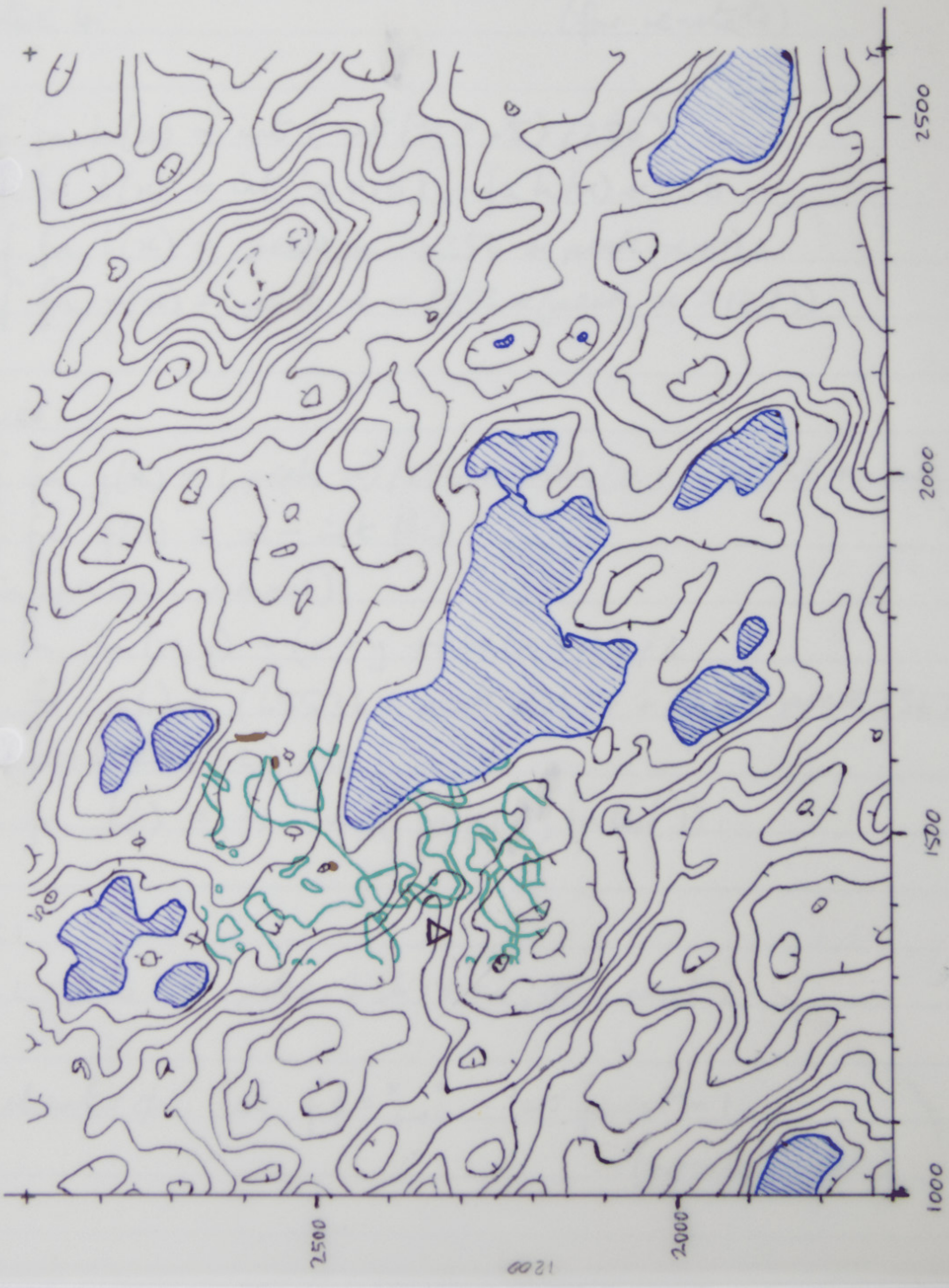
8-3

i	h'_i	i	h'_i	i	h'_i	i	h'_i
0	77	30	123	60	66	90	70
	80		125		67		73
	84		126		69		77
	88		130		71		80
	92		134		73		82
5	96	35	137	65	74	95	85
	101		137		73		88
	104		138		73		90
	108		138		71		93
	112		137		69		95
10	115	40	135	70	66	100	96
	118		133		62		96
	120		129		58		96
	123		123		54		96
	126		118		52		93
15	129	45	111	75	52	105	92
	131		105		54		90
	133		101		55		85
	134		97		58		80
	134		93		59		75
20	133	50	90	80	62	110	71
	133		86		63		67
	131		82		63		63
	130		78		65		60
	129		74		65		58
25	126	55	71	85	65	115	55
	123		69		66		52
	122		67		66		50
	122		67		67		47
	122		67		69		44



8-4

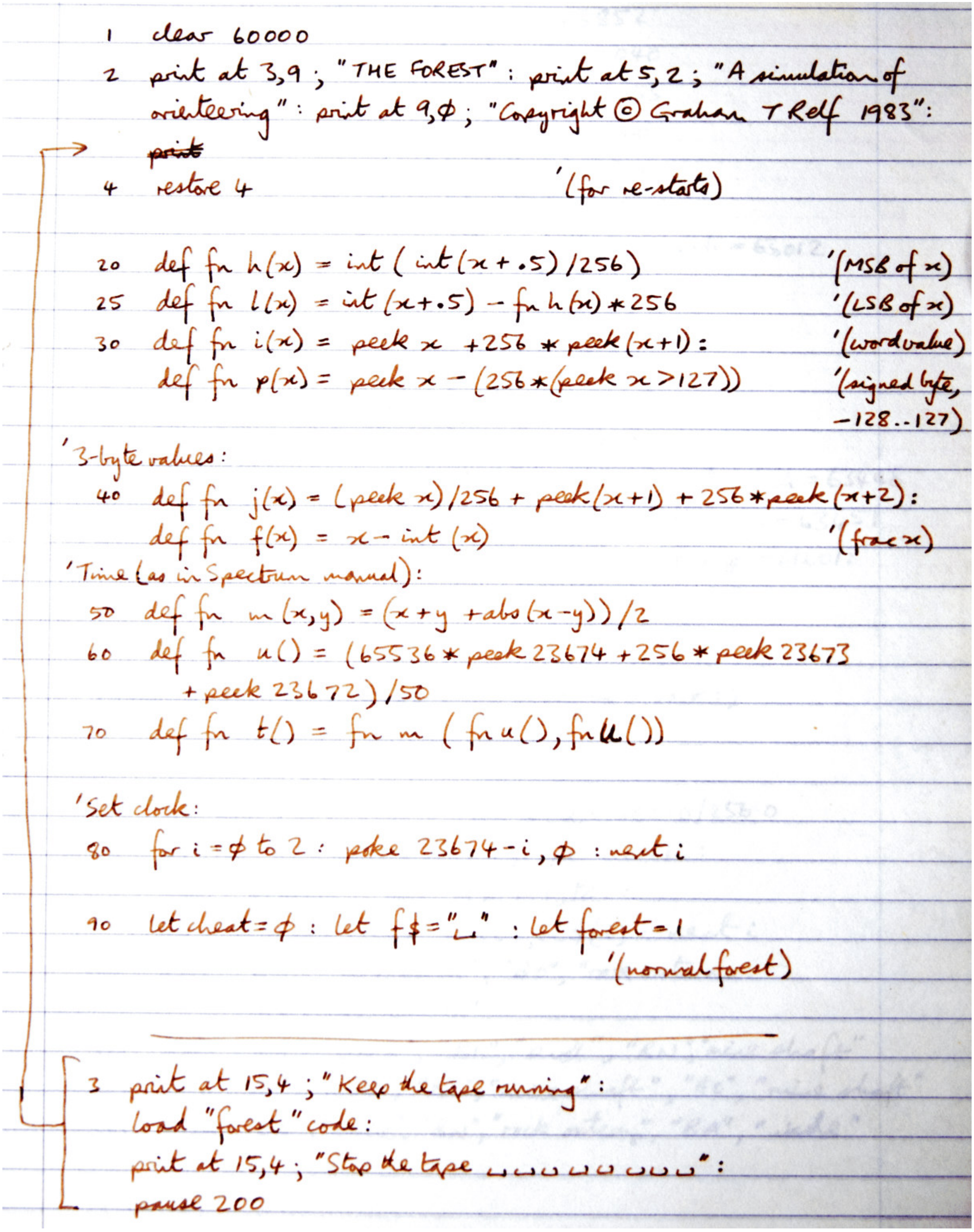
i	h'_i	i	h'_i	i	h'_i	i	h'_i	i	h'_i
120	43	150	24	180	36	210	17	240	84
	41		25		35		15		84
	40		25		35		14		85
	39		26		33		13		85
	36		26		33		11		84
125	35	155	25	185	32	215	11	245	84
	33		25		30		10		82
	32		25		28		10		81
	30		25		25		10		80
	28		25		20		11		75
130	24	160	26	190	15	220	13	250	73
	20		28		11		14		71
	15		29		10		17		71
	11		30		9		20		73
	7		33		9		22		74
135	3	165	36	195	9	225	25	255	75
	2		37		9		28		
	2		39		11		30		
	2		39		14		35		
	2		40		15		39		
140	2	170	40	200	17	230	41		
	2		40		17		45		
	3		39		18		50		
	6		39		18		58		
	7		39		18		63		
145	10	175	37	205	18	235	69		
	11		37		18		73		
	15		37		18		77		
	18		36		17		80		
	22		36		17		82		



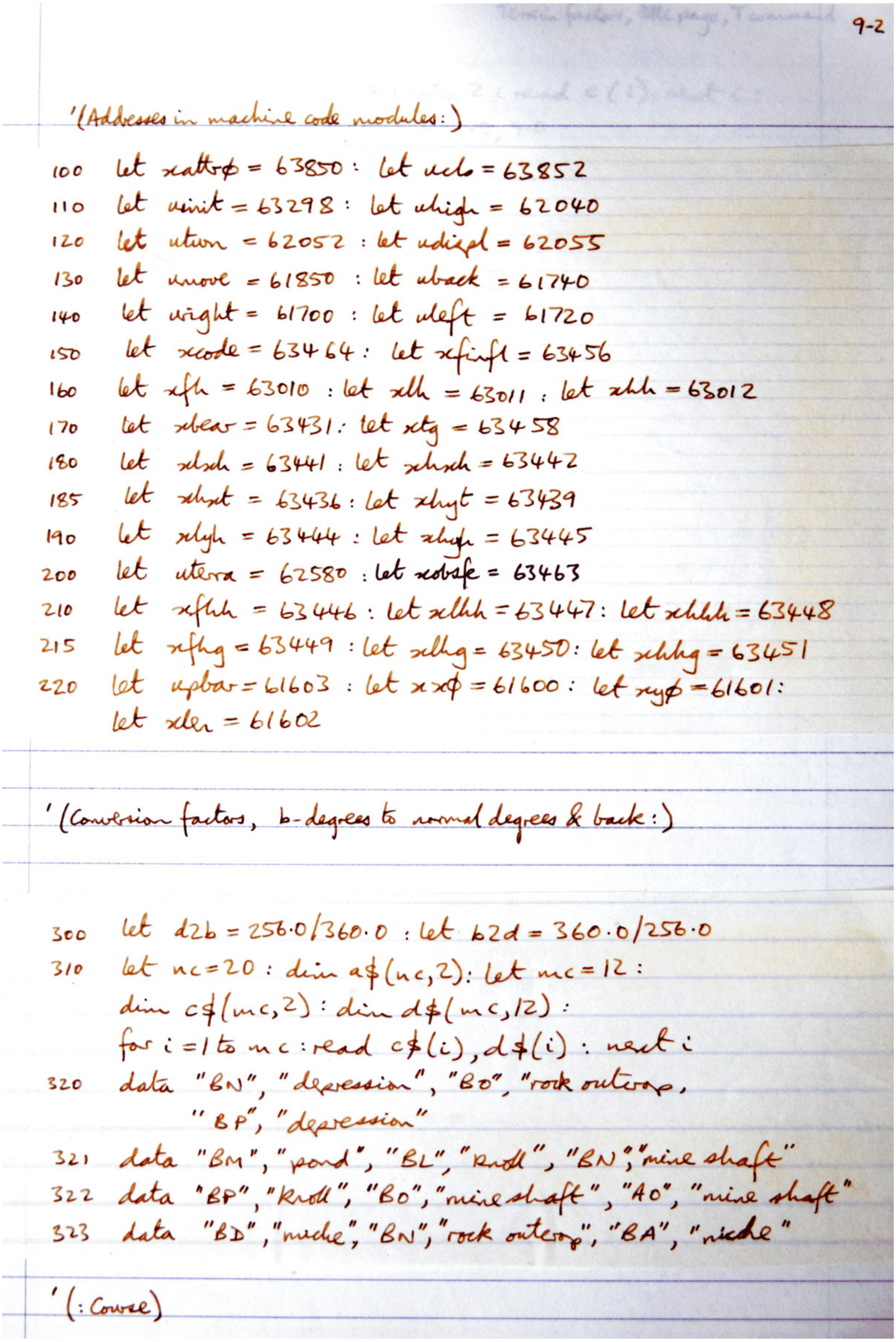
Part of the complex forest (contours, lakes, some vegetation boundaries near the start).

Part 9 - The BASIC shell

I have not retyped this. You might get the digital version from World of Spectrum (the page for The Forest) because they have TAP format files for use with Spectrum emulators. They should enable the BASIC source to be seen once loaded. This would not work for assembly files of course because only the assembled machine code is present. Even if you disassembled that you would not have much clue as to how it works without my comments.

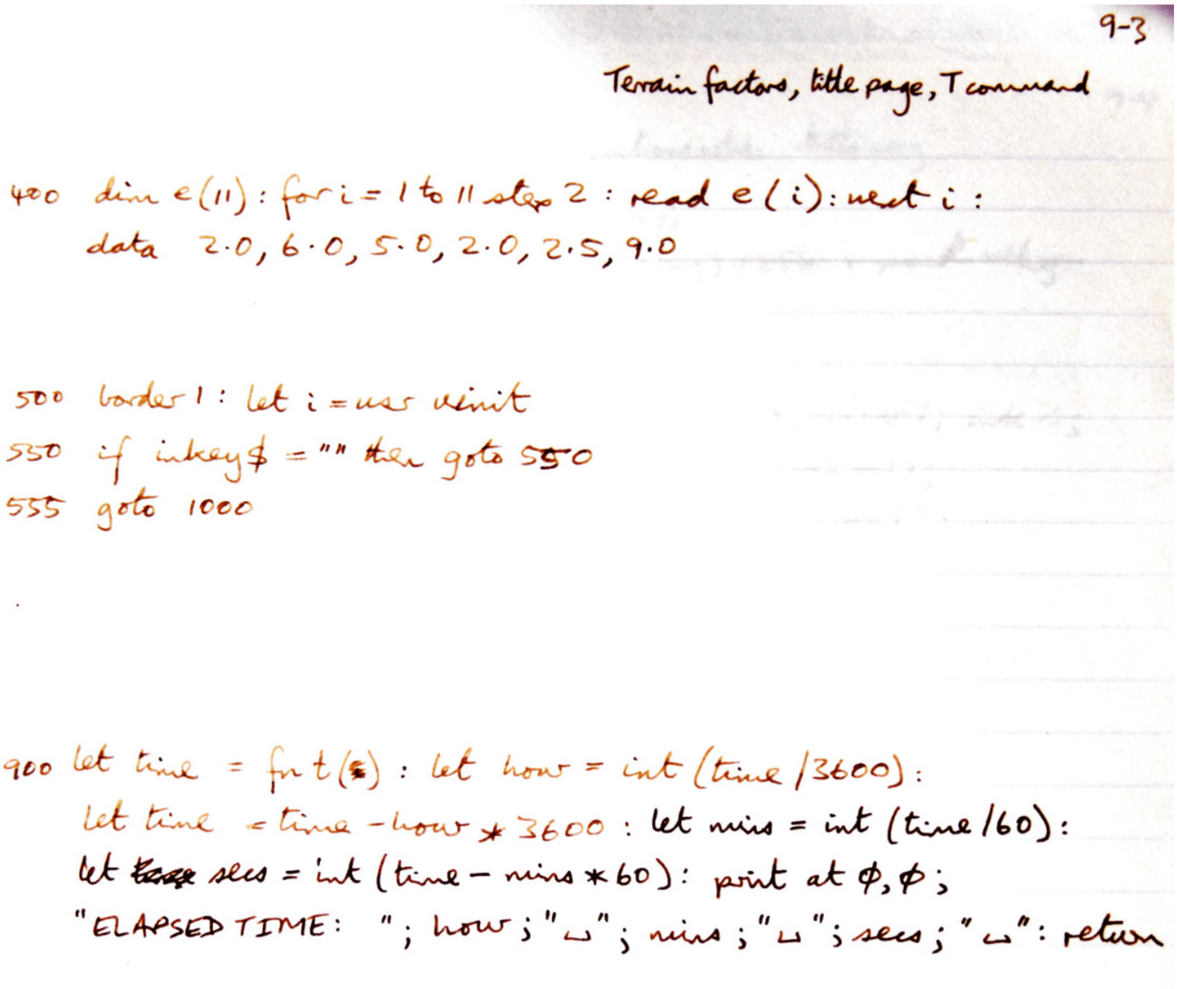


```
1   clear 60000
2   print at 3,9; "THE FOREST": print at 5,2; "A simulation of
    orienteering": print at 9,ø; "Copyright © Graham T Relf 1983":
    ~~print~~
4   restore 4                            '(for re-starts)

20  def fn h(x) = int ( int (x+.5) /256)            '(MSB of x)
25  def fn l(x) = int (x+.5) - fn h(x)*256          '(LSB of x)
30  def fn i(x) = peek x +256 * peek (x+1):         '(word value)
    def fn p(x) = peek x - (256*(peek x >127))      '(signed byte,
                                                    -128..127)
'3-byte values:
40  def fn j(x) = (peek x)/256 + peek(x+1) + 256*peek (x+2):
    def fn f(x) = x - int (x)                       '(frac x)
'Time (as in Spectrum manual):
50  def fn m (x,y) = (x+y +abs (x-y))/2
60  def fn u() = (65536*peek 23674 +256*peek 23673
       + peek 23672)/50
70  def fn t() = fn m ( fn u(), fn u())

'Set clock:
80  for i=ø to 2: poke 23674-i,ø :next i

90  let cheat=ø : let f$="␣" : let forest=1
                                '(normal forest)

3   print at 15,4; "Keep the tape running":
    load "forest" code:
    print at 15,4; "Stop the tape ␣␣␣␣␣␣␣␣":
    pause 200
```



9-2

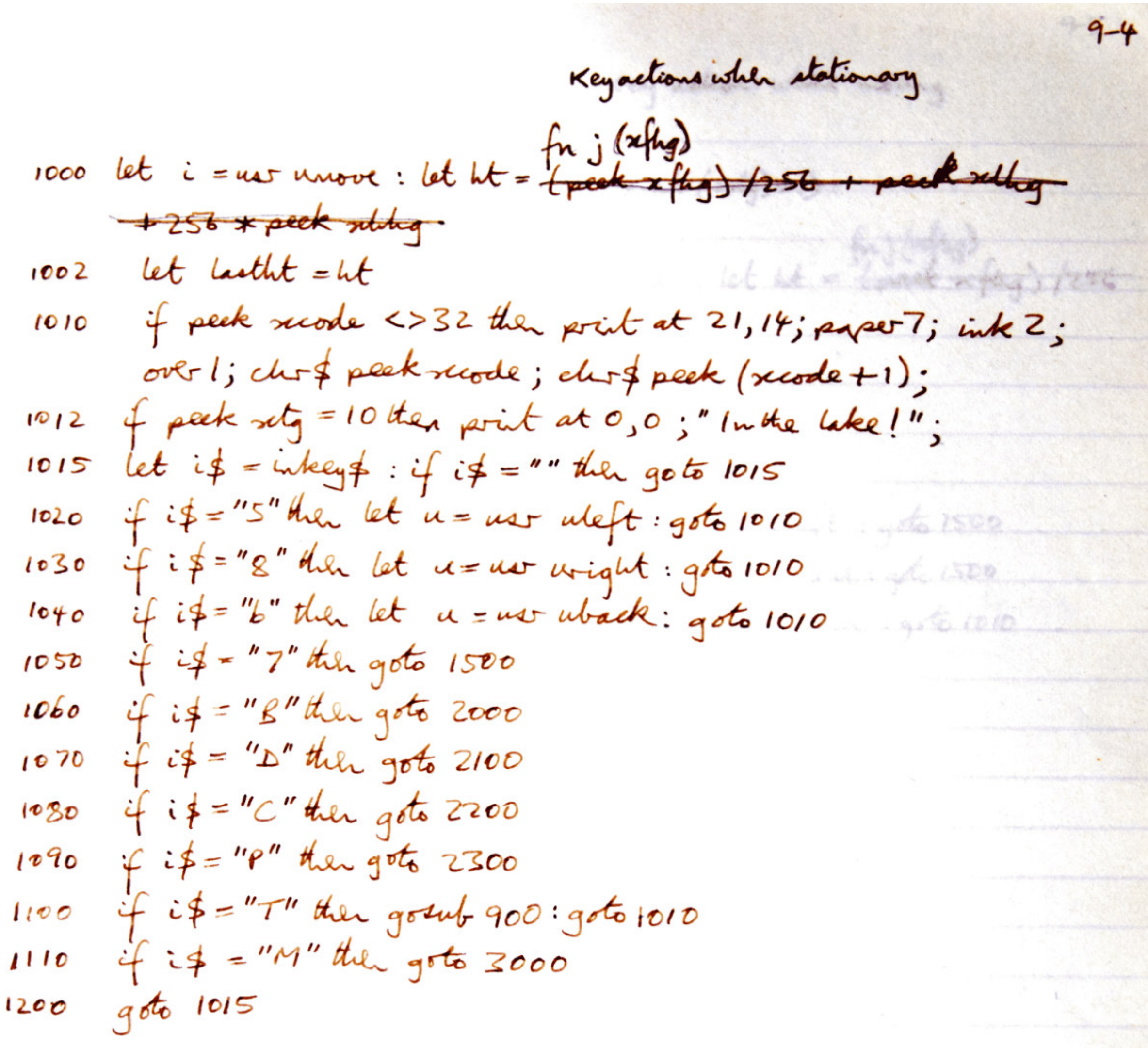
```
'(Addresses in machine code modules:)

100  let xattrφ = 63850 : let ucls = 63852
110  let uinit = 63298 : let uhigh = 62040
120  let uturn = 62052 : let udispl = 62055
130  let umove = 61850 : let uback = 61740
140  let uright = 61700 : let uleft = 61720
150  let xcode = 63464 : let xfinfl = 63456
160  let xfh = 63010 : let xlh = 63011 : let xhh = 63012
170  let xbear = 63431 : let xtg = 63458
180  let xlsch = 63441 : let xhsch = 63442
185  let xhst = 63436 : let xhyt = 63439
190  let xlyh = 63444 : let xhyh = 63445
200  let uterra = 62580 : let xobsfe = 63463
210  let xfhh = 63446 : let xlhh = 63447 : let xhhh = 63448
215  let xfhg = 63449 : let xlhg = 63450 : let xhhg = 63451
220  let upbar = 61603 : let xxφ = 61600 : let xyφ = 61601 :
     let xlen = 61602

'(Conversion factors, b-degrees to normal degrees & back:)

300  let d2b = 256.0/360.0 : let b2d = 360.0/256.0
310  let nc=20 : dim a$(nc,2) : let mc=12 :
     dim c$(mc,2) : dim d$(mc,12) :
     for i=1 to mc : read c$(i), d$(i) : next i
320  data "BN", "depression", "BO", "rock outcrop,
          "BP", "depression"
321  data "BM", "pond", "BL", "knoll", "BN","mine shaft"
322  data "BP", "knoll", "BO", "mine shaft", "AO", "mine shaft"
323  data "BD", "niche", "BN", "rock outcrop", "BA", "niche"

'(:Course)
```



9-3

Terrain factors, title page, T command

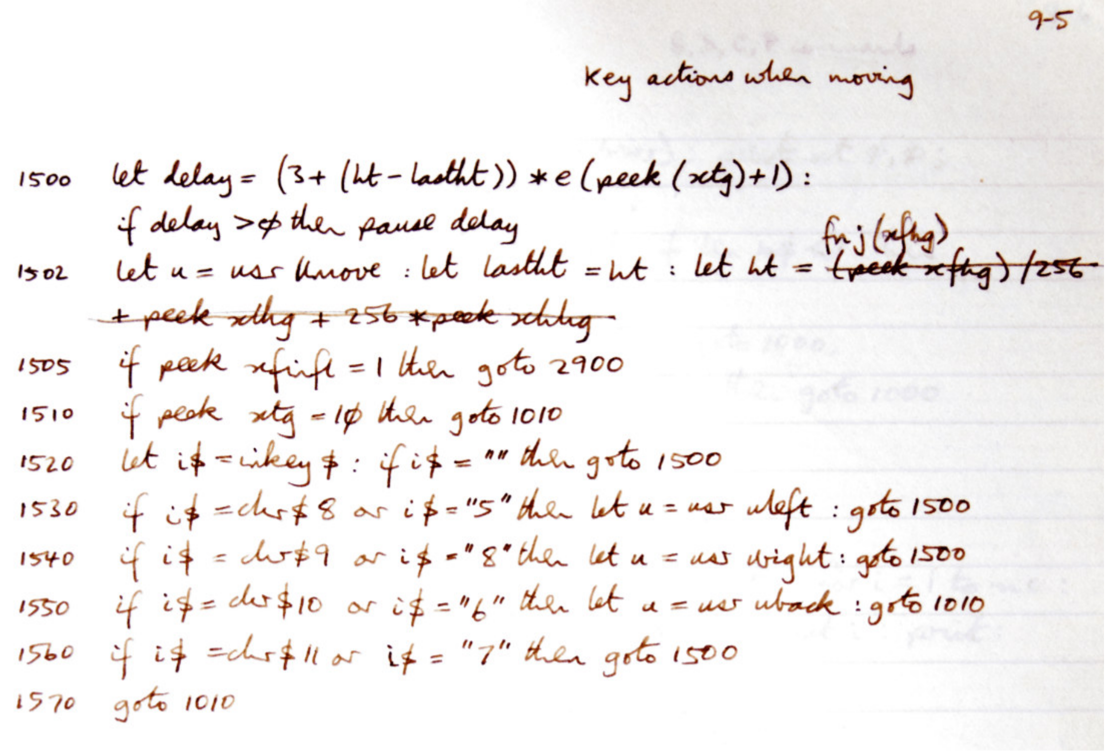
```
400 dim e(11): for i = 1 to 11 step 2 : read e(i): next i :
    data 2.0, 6.0, 5.0, 2.0, 2.5, 9.0


500 border 1 : let i = usr uinit
550 if inkey$ = "" then goto 550
555 goto 1000


900 let time = fn t([illegible]) : let hour = int (time /3600):
    let time = time - hour * 3600 : let mins = int (time /60):
    let ~~[illegible]~~ secs = int (time - mins * 60): print at ɸ,ɸ;
    "ELAPSED TIME: "; hour ;"␣"; mins ;"␣"; secs ; "␣": return
```



9-4

Key actions when stationary

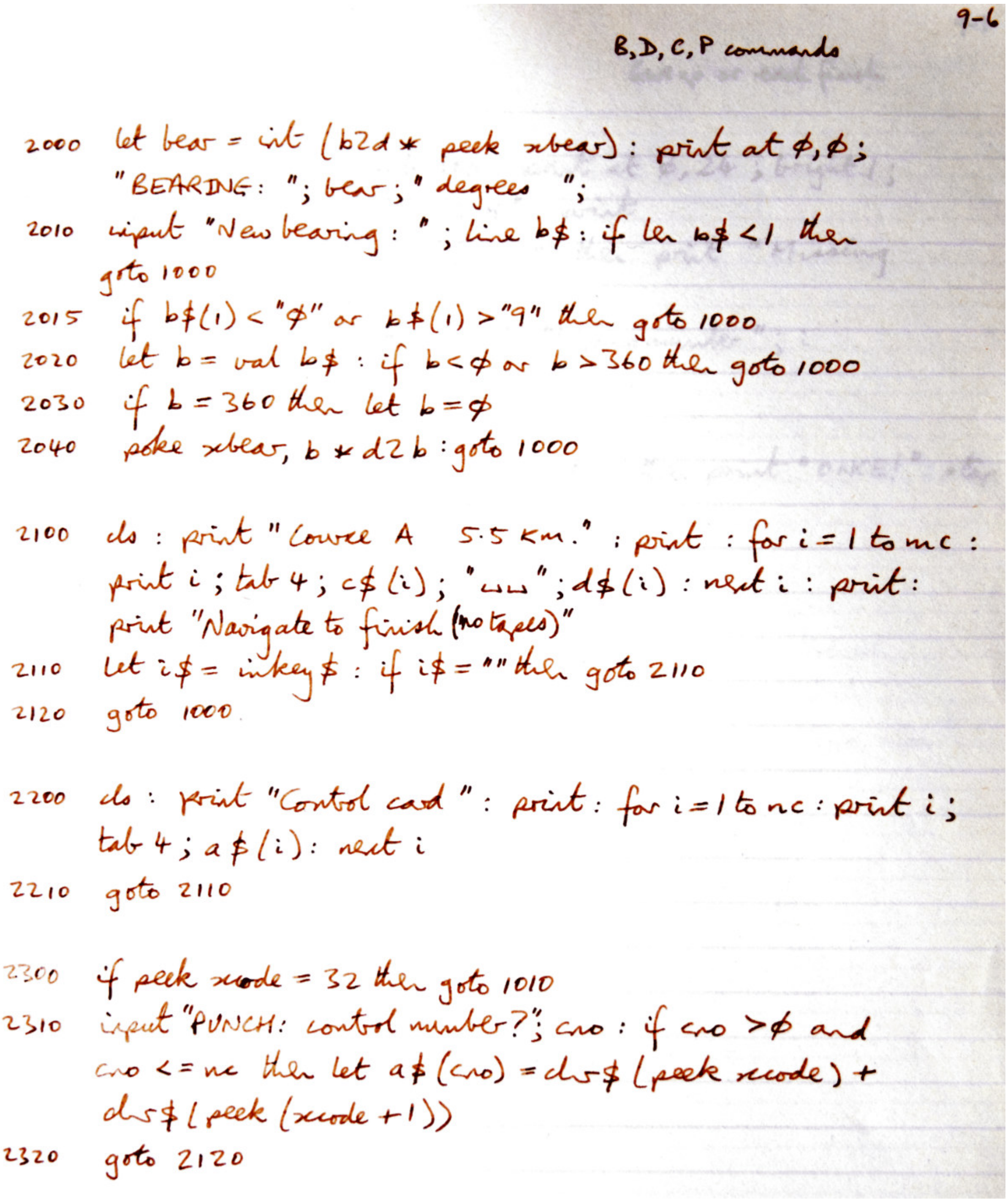
```
1000 let i = usr umove : let ht = fn j (xfhg) ~~(peek xfhg)/256 + peek xdhg
     + 256 * peek xdhg~~
1002  let lastht = ht
1010  if peek xcode <>32 then print at 21,14; paper 7; ink 2;
      over 1; chr$ peek xcode ; chr$ peek (xcode+1);
1012 if peek xtg = 10 then print at 0,0 ;"In the lake!";
1015 let i$ = inkey$ : if i$ = "" then goto 1015
1020 if i$ = "5" then let u = usr uleft : goto 1010
1030 if i$ = "8" then let u = usr uright : goto 1010
1040  if i$ = "6" then let u = usr uback : goto 1010
1050  if i$ = "7" then goto 1500
1060  if i$ = "S" then goto 2000
1070 if i$ = "D" then goto 2100
1080 if i$ = "C" then goto 2200
1090 if i$ = "P" then goto 2300
1100 if i$ = "T" then gosub 900 : goto 1010
1110 if i$ = "M" then goto 3000
1200 goto 1015
```



9-5

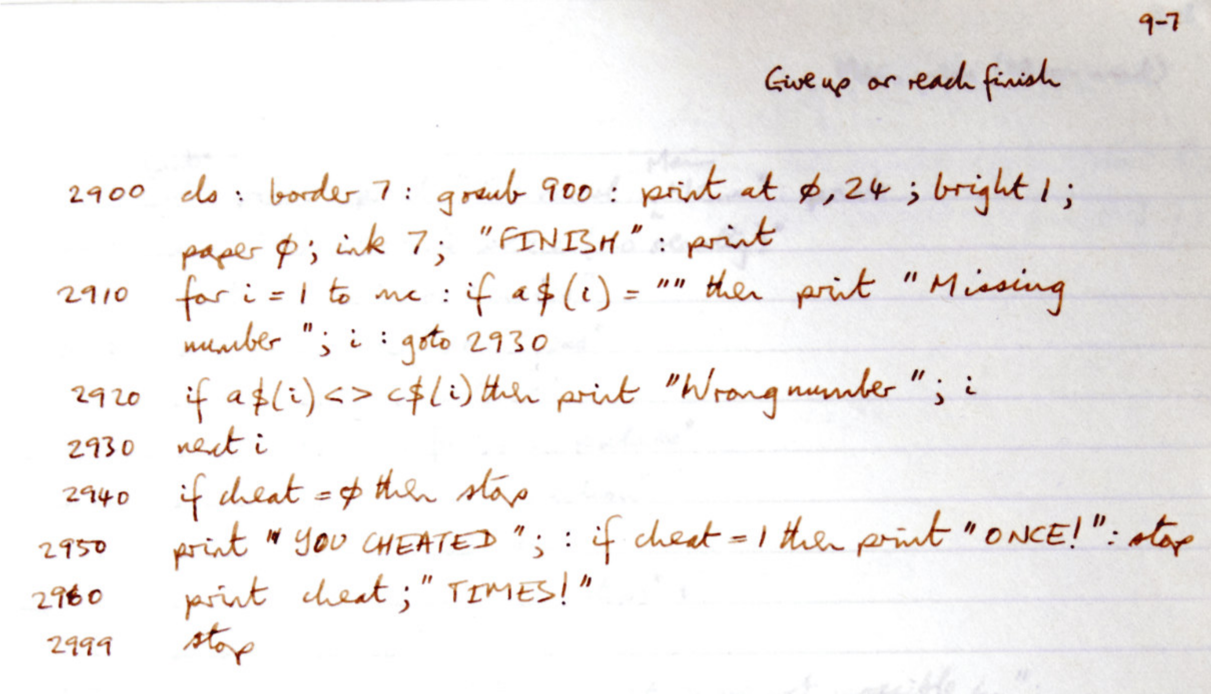
Key actions when moving

```
1500 let delay = (3+ (ht - lastht)) * e (peek (xtg)+1) :
     if delay >ø then pause delay
1502 let u = usr umove : let lastht = ht : let ht = fnj (xfhg) ~~(peek xfhg)/256
     + peek xdhg + 256 * peek xchhg~~
1505 if peek xfinfl = 1 then goto 2900
1510 if peek xtg = 1ø then goto 1010
1520 let i$ = inkey$ : if i$ = "" then goto 1500
1530 if i$ = chr$ 8 or i$ = "5" then let u = usr uleft : goto 1500
1540 if i$ = chr$ 9 or i$ = "8" then let u = usr uright : goto 1500
1550 if i$ = chr$ 10 or i$ = "6" then let u = usr uback : goto 1010
1560 if i$ = chr$ 11 or i$ = "7" then goto 1500
1570 goto 1010
```



B,D,C,P commands

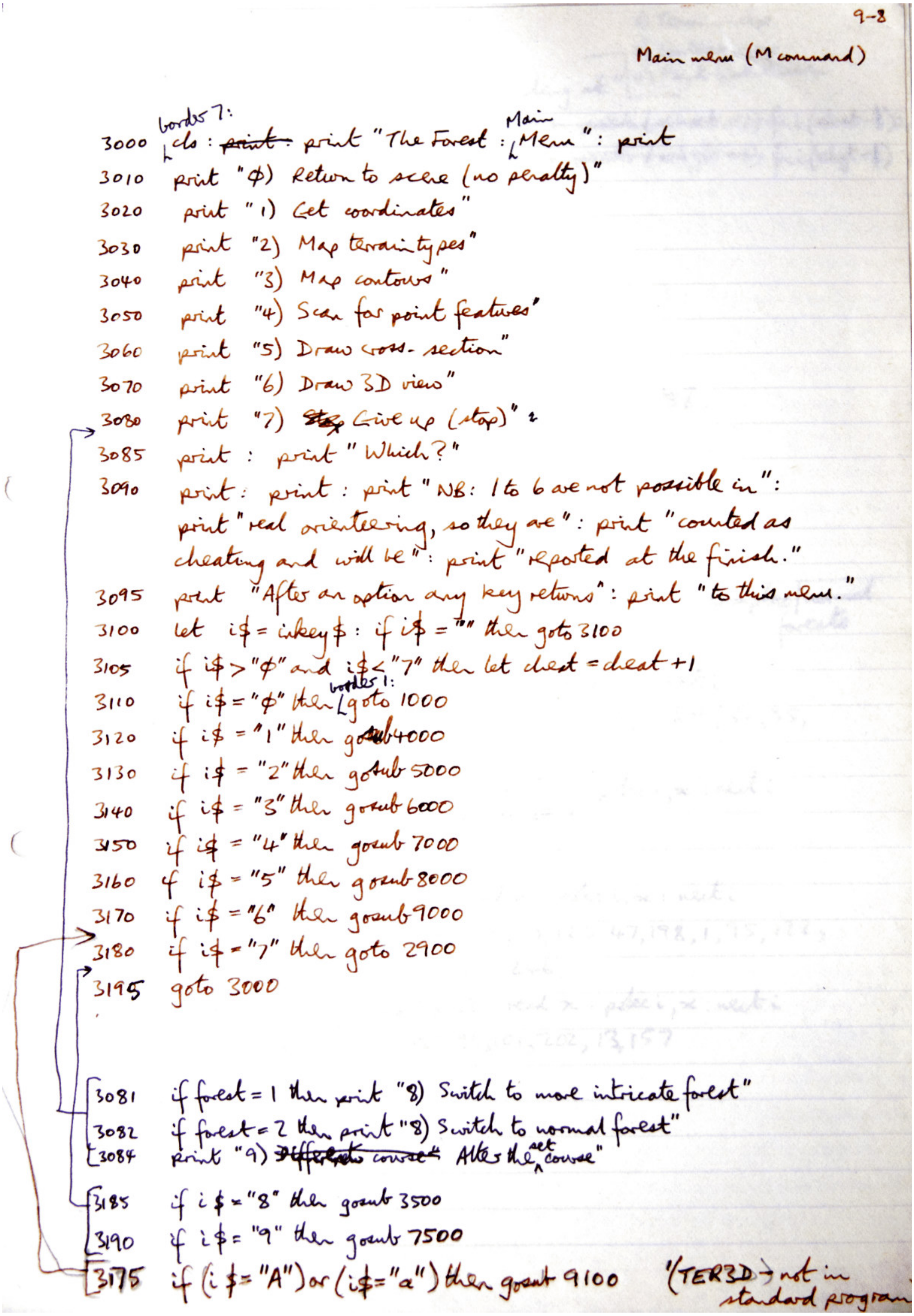
```
2000  let bear = int (b2d * peek xbear) : print at ø,ø;
      "BEARING: "; bear ;" degrees  ";
2010  input "New bearing: "; line b$ : if len b$<1 then
      goto 1000
2015  if b$(1) < "ø" or b$(1) >"9" then goto 1000
2020  let b = val b$ : if b<ø or b>360 then goto 1000
2030  if b=360 then let b=ø
2040  poke xbear, b * d2b : goto 1000

2100  cls : print "Course A   5.5 km." : print : for i=1 to nc :
      print i ; tab 4; c$ (i); "␣␣"; d$(i) : next i : print:
      print "Navigate to finish (no tapes)"
2110  let i$ = inkey$ : if i$ ="" then goto 2110
2120  goto 1000

2200  cls : print "Control card" : print : for i=1 to nc : print i;
      tab 4 ; a$ (i) : next i
2210  goto 2110

2300  if peek xcode = 32 then goto 1010
2310  input "PUNCH: control number?"; cno : if cno >ø and
      cno <= nc then let a$ (cno) = chr$ (peek xcode) +
      chr$ (peek (xcode +1))
2320  goto 2120
```



9-7

Give up or reach finish

```
2900 cls : border 7 : gosub 900 : print at ø,24 ; bright 1;
     paper ø; ink 7; "FINISH" : print
2910 for i = 1 to mc : if a$(i) = "" then print "Missing
     number "; i : goto 2930
2920 if a$(i) <> c$(i) then print "Wrongnumber "; i
2930 next i
2940 if cheat = ø then stop
2950 print "YOU CHEATED "; : if cheat = 1 then print "ONCE!": stop
2960 print cheat; " TIMES!"
2999 stop
```



Main menu (M command)

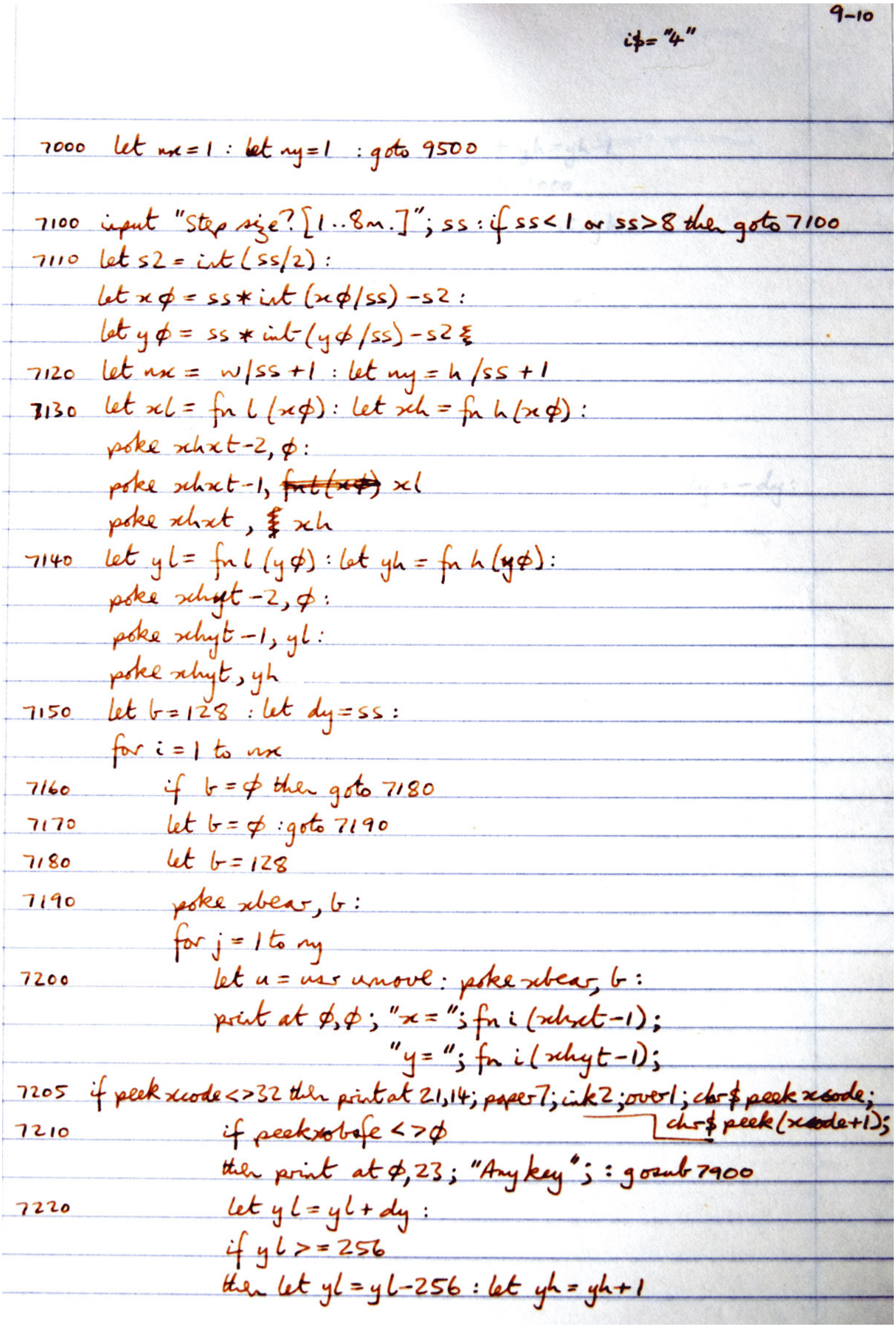
```
3000 border 7: cls : ~~print~~ print "The Forest : Main Menu": print
3010 print "Ø) Return to scene (no penalty)"
3020 print "1) Get coordinates"
3030 print "2) Map terrain types"
3040 print "3) Map contours"
3050 print "4) Scan for point features'
3060 print "5) Draw cross- section"
3070 print "6) Draw 3D view"
3080 print "7) ~~Stop~~ Give up (stop)":
3085 print : print "Which?"
3090 print: print : print "NB: 1 to 6 are not possible in":
     print "real orienteering, so they are": print "counted as
     cheating and will be": print "reported at the finish."
3095 print "After an option any key returns": print "to this menu."
3100 let i$ = inkey$ : if i$ = "" then goto 3100
3105 if i$>"Ø" and i$<"7" then let cheat = cheat +1
3110 if i$ = "Ø" then border 1: goto 1000
3120 if i$ = "1" then gosub 4000
3130 if i$ = "2" then gosub 5000
3140 if i$ = "3" then gosub 6000
3150 if i$ = "4" then gosub 7000
3160 if i$ = "5" then gosub 8000
3170 if i$ = "6" then gosub 9000
3180 if i$ = "7" then goto 2900
3195 goto 3000


3081 if forest = 1 then print "8) Switch to more intricate forest"
3082 if forest = 2 then print "8) Switch to normal forest"
3084 print "9) ~~Different course~~ Alter the set course"
3185 if i$ = "8" then gosub 3500
3190 if i$ = "9" then gosub 7500
3175 if (i$="A") or (i$="a") then gosub 9100    '(TER3D → not in standard program
```

1) Get coordinates
2) Terrain map
3) Contour map
4) Point feature scan

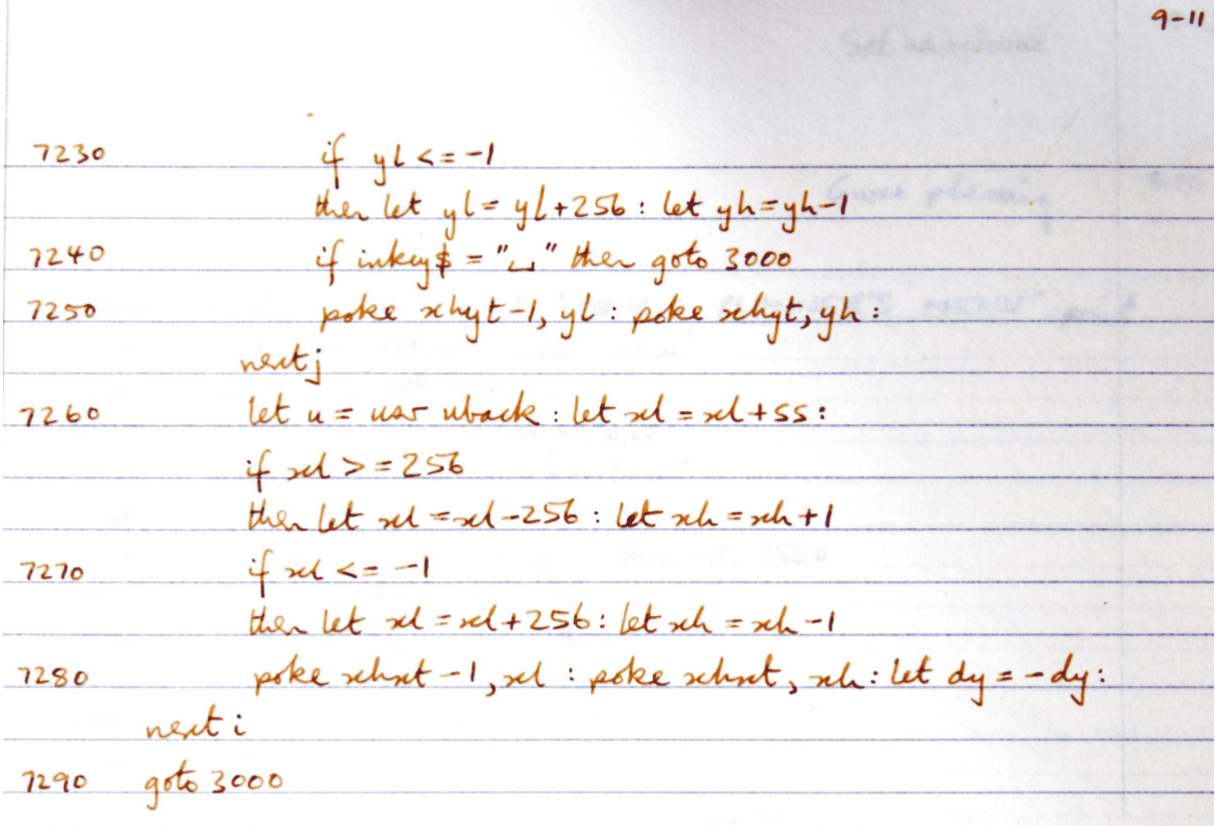
```
4000 cls: print : print "You are standing at"
4010 print "x = "; ~~256 * peek xhxt + peek (xhxt -1)~~ fn i (xhxt-1):
~~4020~~ print "y = "; ~~256* peek xhyt + peek (xhyt -1)~~ fn i(xhyt-1)
4030 if inkey$ = ""then goto 4030
4040 return

5000 let nx =32 : let ny =24 : poke 61322,13 :
     poke 61323,239 : goto 9500

6000 let nx =32 : let ny =24 : poke 61322,187 :
     poke 61323,238 : goto 9500

7000 let nx =1 : let ny=1 : goto 9500
```

Complex/normal forests

```
3500 if forest=2 then goto 3700
3510 let forest =2 : restore 3510
3520 for i = 62783 to 62802 : read x : poke i,x : next i
3530 data 38,ϕ,1,84,236,9,78,6,ϕ,42,35,246,9,34,35,
                60500
          246,193,16,207,201
3540 ~~return~~ for i = 63013 to 63022 : read x : poke i,x : next i
3560 data ϕ,27,13,26,21,21,22,11,29,1
3590 return
3700 let forest =1 : restore 3700
3710 for i = 62783 to 62802 : read x : poke i,x : next i
3720 data 85,95,122,254,127,56,10,123,47,198,1,95,122,
          47,206,0,87,42,34,246
3730 ~~return~~ for i = 63013 to 63022 : read x : poke i,x : next i
3750 data 200,3,165,37,127,96,101,202,13,157
3790 return
```

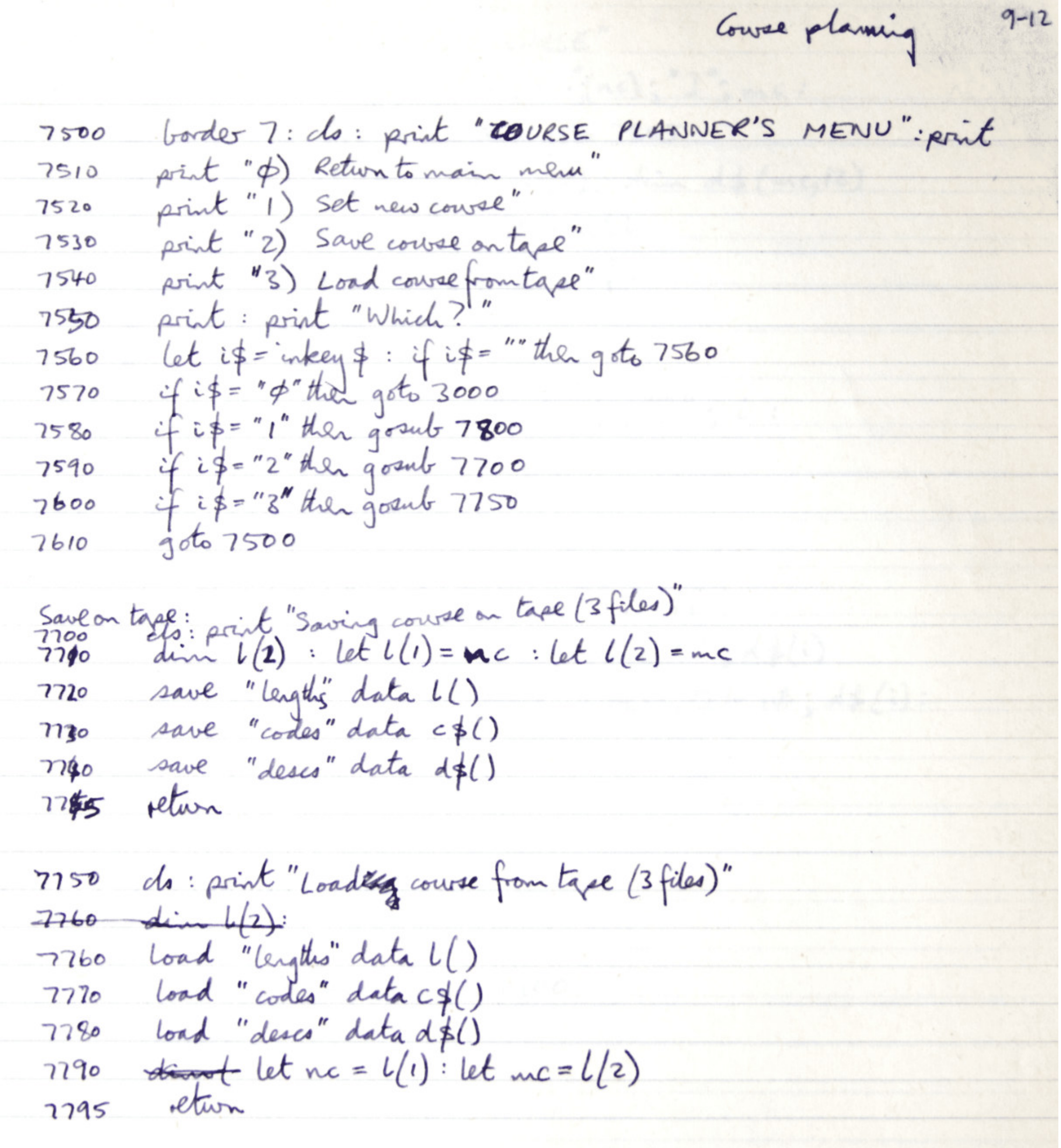


```
                                                   9-10
                                         i$="4"

7000  let nx=1 : let ny=1  : goto 9500

7100  input "Step size? [1..8m.]"; ss : if ss<1 or ss>8 then goto 7100
7110  let s2 = int (ss/2) :
      let xØ = ss*int (xØ/ss) -s2 :
      let yØ = ss * int (yØ/ss) -s2
7120  let nx =  w/ss +1 : let ny = h /ss +1
7130  let xl = fn l (xØ): let xh = fn h(xØ) :
       poke xhxt-2, Ø:
       poke xhxt-1, ~~fnl(xØ)~~ xl
       poke xhxt , xh
7140  let yl= fn l (yØ) : let yh = fn h(yØ):
       poke xhyt -2, Ø :
       poke xhyt -1, yl :
       poke xhyt, yh
7150   let b=128  : let dy=ss :
       for i=1 to nx
7160        if  b=Ø then goto 7180
7170        let b=Ø :goto 7190
7180        let b=128
7190        poke xbear, b :
            for j=1 to ny
7200           let u = usr umove : poke xbear, b :
               print at Ø,Ø ; "x="; fn i (xhxt-1);
                                  "y="; fn i(xhyt-1);
7205 if peek xcode<>32 then print at 21,14; paper7; ink2 ;over1; chr$ peek xcode;
7210           if peek xobufe <>Ø                        chr$ peek (xcode+1);
               then print at Ø,23; "Any key"; : gosub 7900
7220           let yl=yl+dy :
               if yl>=256
               then let yl=yl-256 : let yh=yh+1
```



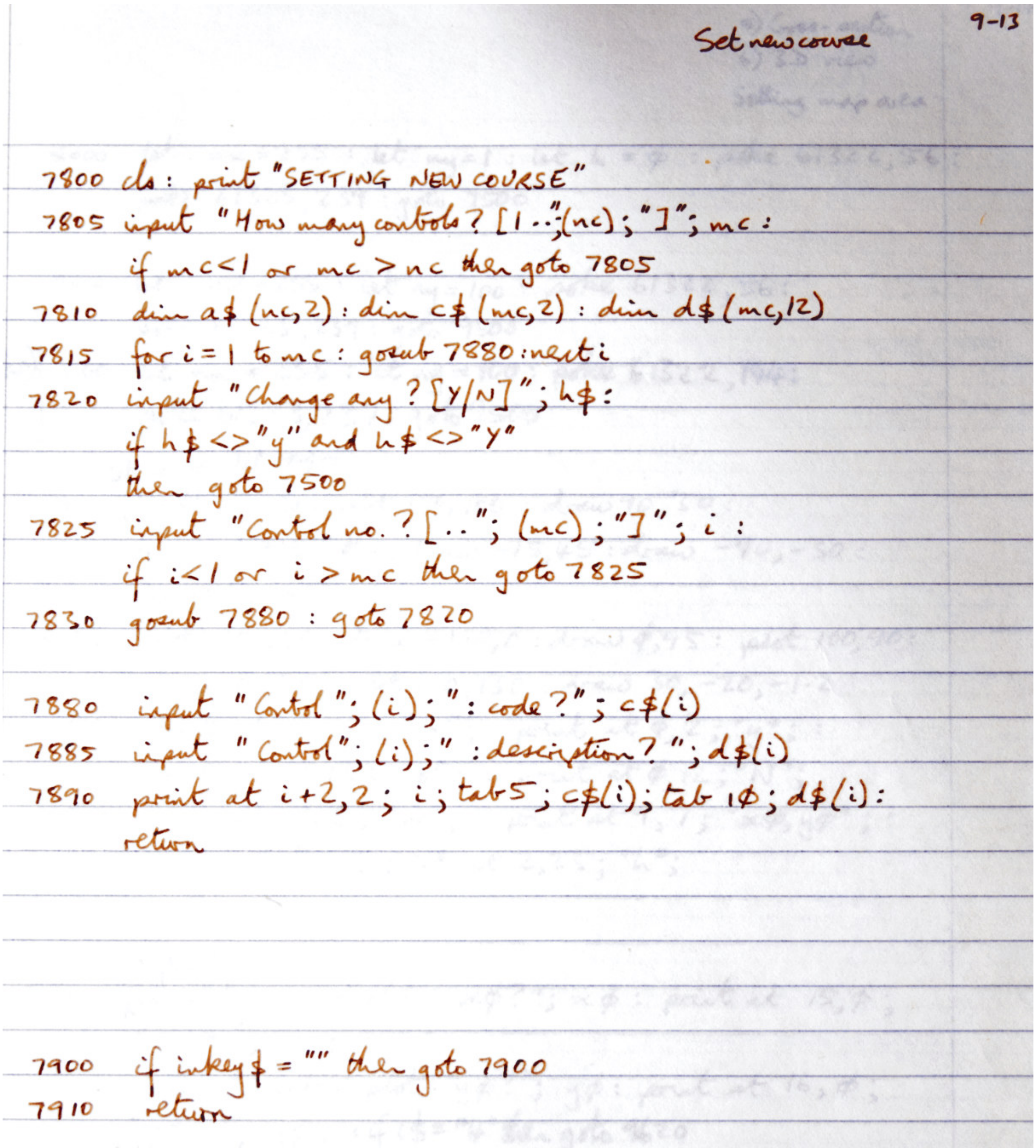
9-11

```
7230            if yl<=-1
                then let yl=yl+256 : let yh=yh-1
7240            if inkey$ = "␣" then goto 3000
7250            poke xhyt-1, yl : poke xhyt, yh :
            next j
7260        let u = usr uback : let xl = xl+ss :
            if xl >= 256
            then let xl = xl-256 : let xh = xh+1
7270        if xl <= -1
            then let xl = xl+256 : let xh = xh-1
7280        poke xhxt-1, xl : poke xhxt, xh : let dy = -dy :
        next i
7290    goto 3000
```



Course planning 9-12

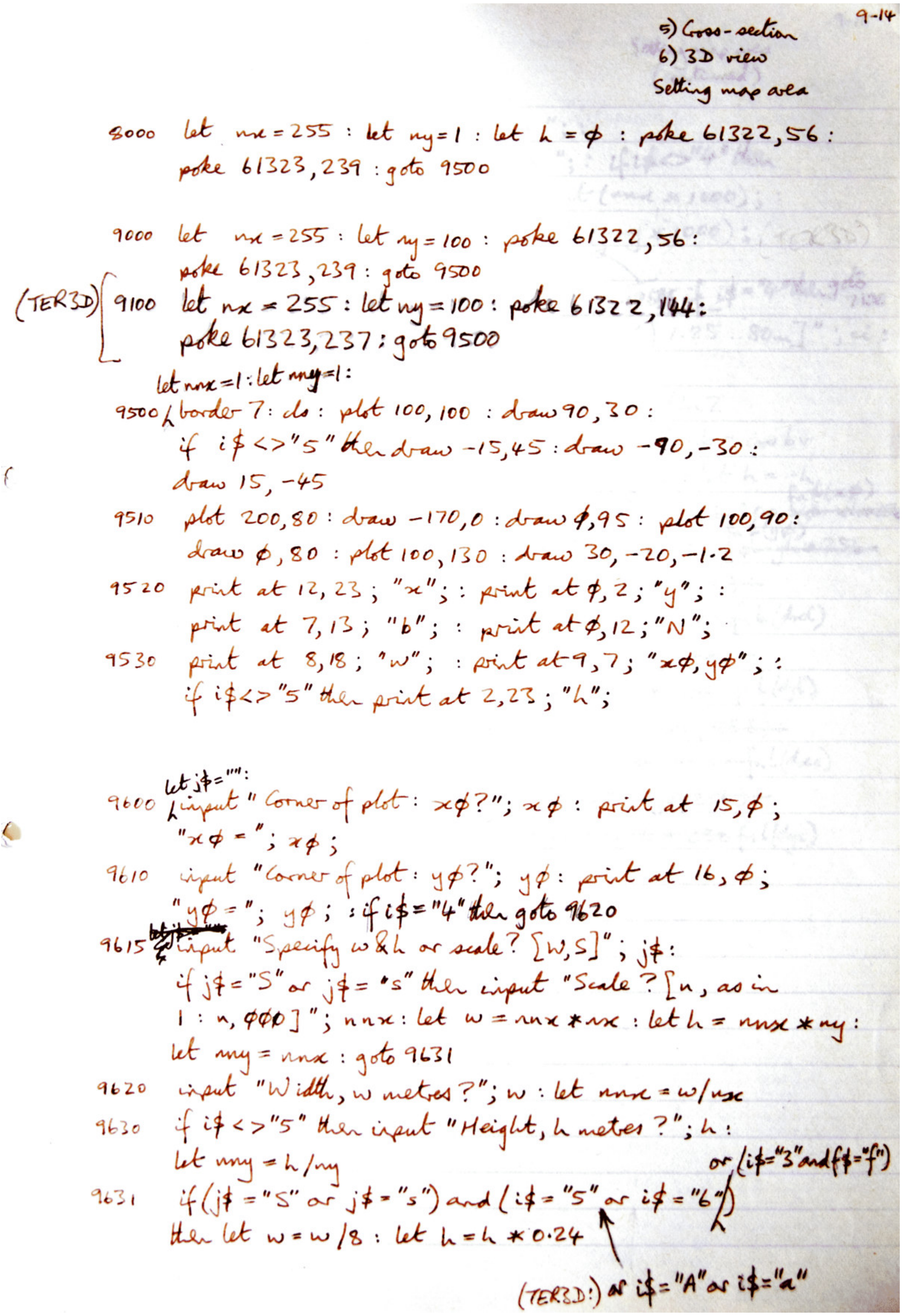
```
7500   border 7: cls: print "COURSE PLANNER'S MENU": print
7510   print "ϕ) Return to main menu"
7520   print "1) Set new course"
7530   print "2) Save course on tape"
7540   print "3) Load course from tape"
7550   print : print "Which?"
7560   let i$=inkey$ : if i$="" then goto 7560
7570   if i$="ϕ" then goto 3000
7580   if i$="1" then gosub 7800
7590   if i$="2" then gosub 7700
7600   if i$="3" then gosub 7750
7610   goto 7500

Save on tape:
7700   cls: print "Saving course on tape (3 files)"
7710   dim l(2) : let l(1)=nc : let l(2)=mc
7720   save "lengths" data l()
7730   save "codes" data c$()
7740   save "descs" data d$()
7745   return

7750   cls : print "Loading course from tape (3 files)"
~~7760   dim l(2)~~:
7760   load "lengths" data l()
7770   load "codes" data c$()
7780   load "descs" data d$()
7790   ~~dim st~~ let nc = l(1) : let mc = l(2)
7795   return
```



9-13

Set new course

```
7800 cls: print "SETTING NEW COURSE"
7805 input "How many controls? [1..";(nc);"]"; mc:
     if mc<1 or mc>nc then goto 7805
7810 dim a$(nc,2): dim c$(mc,2): dim d$(mc,12)
7815 for i=1 to mc: gosub 7880: next i
7820 input "Change any? [Y/N]"; h$:
     if h$<>"y" and h$<>"Y"
     then goto 7500
7825 input "Control no.? [..";(mc);"]"; i :
     if i<1 or i>mc then goto 7825
7830 gosub 7880 : goto 7820

7880 input "Control ";(i);": code?"; c$(i)
7885 input "Control";(i);" :description? "; d$(i)
7890 print at i+2,2; i; tab5; c$(i); tab 1Ø; d$(i):
     return

7900 if inkey$ = "" then goto 7900
7910 return
```



5) Cross-section
6) 3D view
Setting map area

```
8000  let nx=255 : let ny=1 : let h=ϕ : poke 61322,56 :
      poke 61323,239 : goto 9500

9000  let nx=255 : let ny=100 : poke 61322,56 :
      poke 61323,239 : goto 9500
(TER3D) 9100  let nx=255 : let ny=100 : poke 61322,144:
      poke 61323,237 : goto 9500
   let nnx=1:let nny=1:
9500  border 7 : cls : plot 100,100 : draw 90,30 :
      if i$<>"5" then draw -15,45 : draw -90,-30 :
      draw 15,-45
9510  plot 200,80 : draw -170,0 : draw ϕ,95 : plot 100,90:
      draw ϕ,80 : plot 100,130 : draw 30,-20,-1.2
9520  print at 12,23; "x"; : print at ϕ,2; "y"; :
      print at 7,13; "b"; : print at ϕ,12; "N";
9530  print at 8,18; "w"; : print at 9,7; "xϕ,yϕ"; :
      if i$<>"5" then print at 2,23; "h";

     let j$="":
9600  input "Corner of plot : xϕ?"; xϕ : print at 15,ϕ;
      "xϕ ="; xϕ;
9610  input "Corner of plot : yϕ?"; yϕ : print at 16,ϕ;
      "yϕ ="; yϕ; : if i$="4" then goto 9620
9615  ~~let j$~~ input "Specify w&h or scale? [W,S]"; j$:
      if j$="S" or j$="s" then input "Scale ? [n, as in
      1 : n,ϕϕϕ]"; nnx : let w = nnx*nx : let h = nnx*ny :
      let nny = nnx : goto 9631
9620  input "Width, w metres?"; w : let nnx = w/nx
9630  if i$<>"5" then input "Height, h metres ?"; h :
      let nny = h/ny
                                          or (i$="3"and f$="f")
9631  if (j$="S" or j$="s") and (i$="5" or i$="6")
      then let w=w/8 : let h=h*0.24

                              (TER3D!) or i$="A"or i$="a"
```

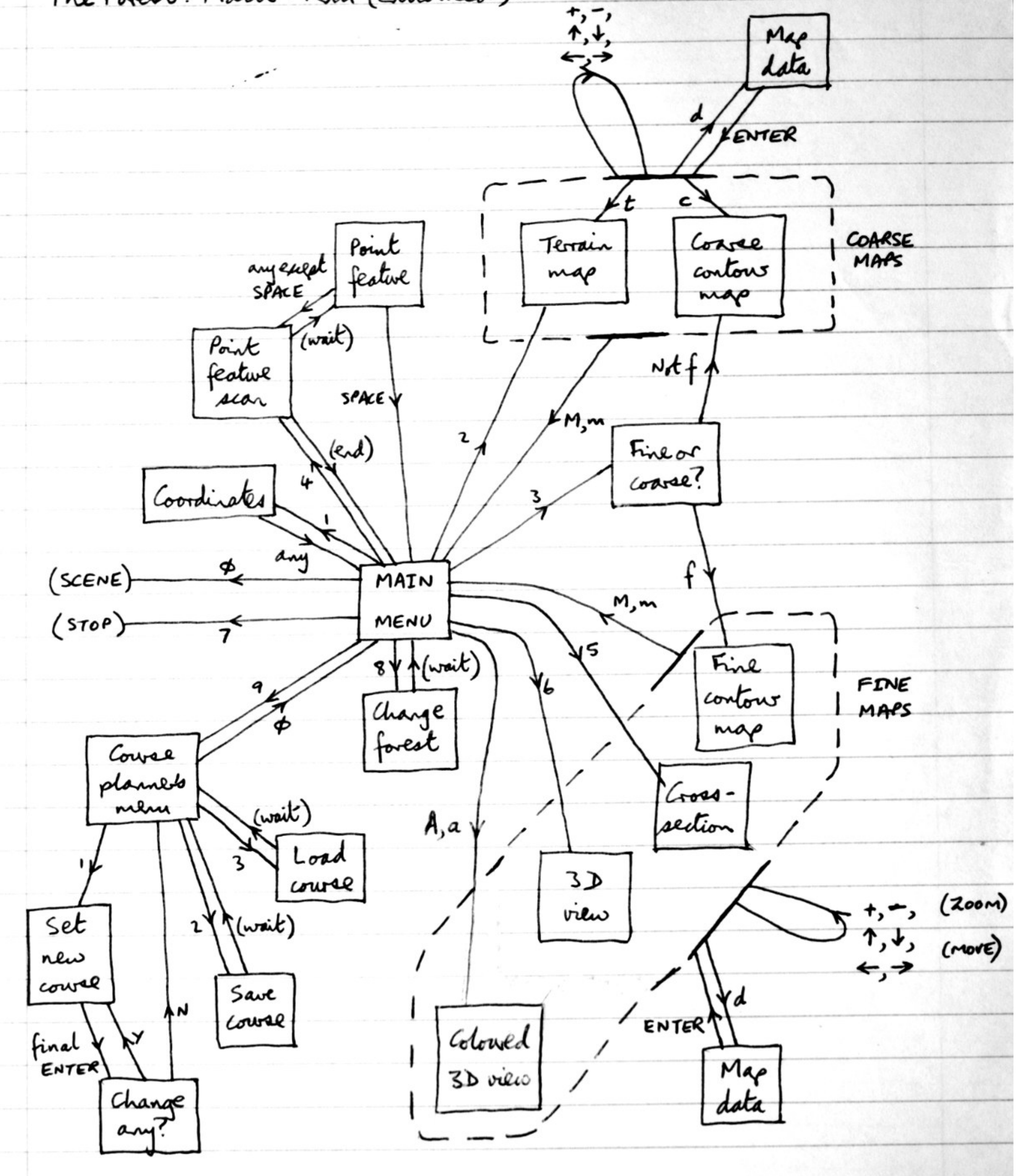
Setting map area
(continued)

```
9632 print at 15,16;"w = ";w;"m.";:
     print at 16,16;"h = ";h;"m.";: if i$<>"4" then
     print at 17,ø;"x scale = 1:"; int (nnx * 1000);:
     print at 18,ø;"y scale = 1:"; int (nny * 1000);
9640 input "Angle b degrees?"; b :
     print at 19,ø;"b = ";b;" degrees"; ~~inv~~
     9635 if i$="4" then goto 7100
9642 if (i$="3" and f$<>"f") then input "Contour interval? [1.25..80m]"; ci:
     if ci<1.25 or ci>80 then goto 9642
9643 if (i$="3" and f$<>"f") then poke 61114, ln (6.4 * ci)/ln2
9645 let br = b* pi /180 : let sb = sin br : let cb = cos br
9647 let xø = xø - h*cb : let yø = yø + h*sb : let h = -h
9650 ~~let xh = int (xø/256):~~ poke 61301, ~~xh~~ fnh(xø): poke 61300, ~~xø-xh*256~~ fnl(xø)
9660 ~~let yh = int (yø/256):~~ poke 61303, ~~yh~~ fnh(yø): poke 61302, ~~yø-yh*256~~ fnl(yø)
9680 let dxl = w *sb/nx : ~~let dxlh = int (dxl/256):~~
     poke 61309, ~~dxlh~~ fnh(dxl): poke 61308, ~~dxl-dxlh*256~~ fnl(dxl)
9690 let dyl = w * cb /nx : ~~let dylh = int (dyl/256):~~
     poke 61311, ~~dylh~~ fnh(dyl) : poke 61310, ~~dyl-dylh*256~~ fnl (dyl)
9700 let dxc =-h *cb/ny : ~~let dxch = int (dxc/256):~~
     poke 61313, ~~dxch~~ fnh(dxc): poke 61312, ~~dxc-dxch*256~~ fnl(dxc)
9710 let dyc = h*sb/ny : ~~let dych = int (dyc/256):~~
     poke 61315, ~~dych~~ fnh(dyc): poke 61314, ~~dyc-dych*256~~ fnl(dyc)
9720 poke 61316, nx : poke 61317, ny
9730 cls : let u = usr 61324
9740 poke 61348,191
9900 if inkey$ ="" then goto 9900
9999 return
```

Map enhancements : re-scale, scroll, switch type, get data

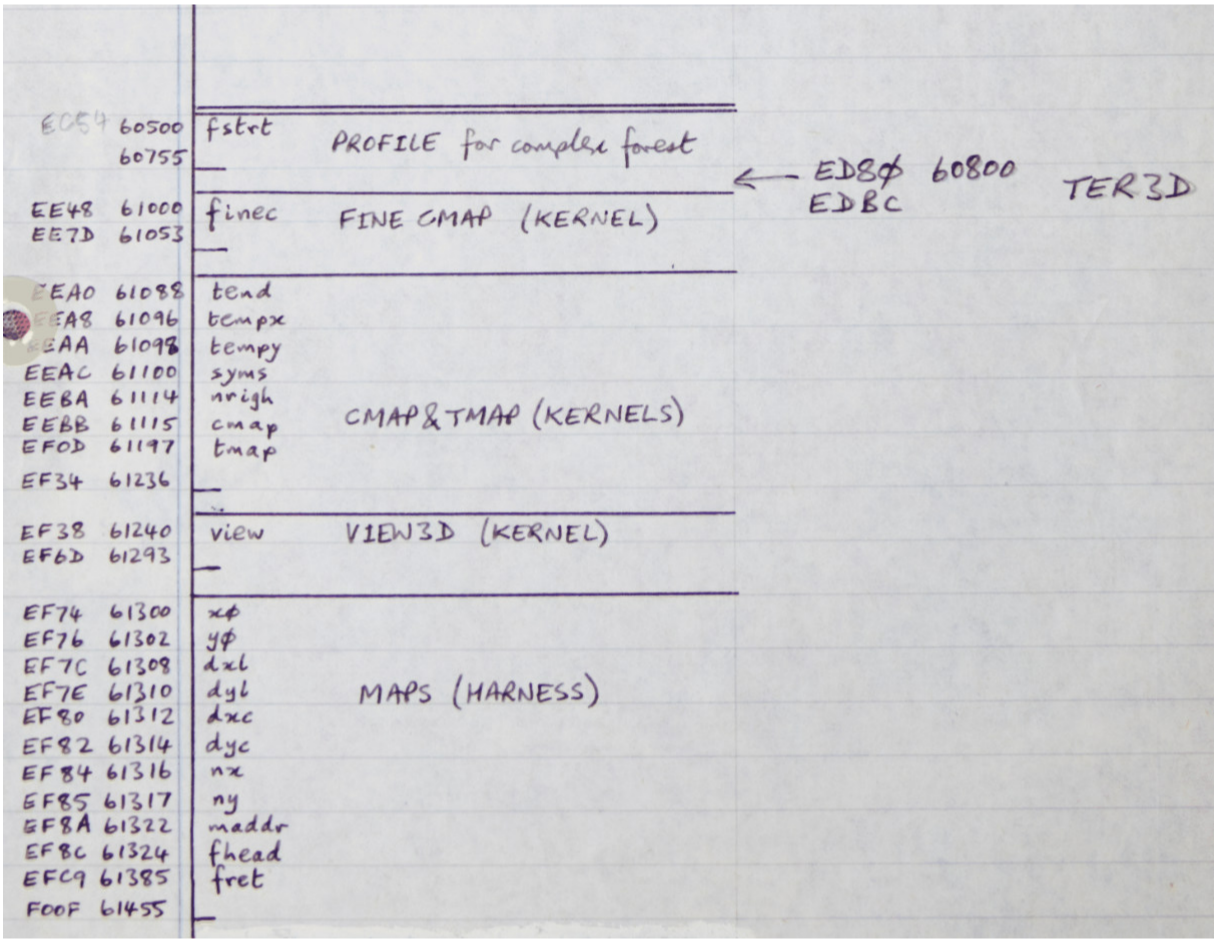
```
                                                   ci must be
                   let                                 defined
9641   pause 100 : cls : ci = 5          (cls needed for "fine" maps,)
9730   let  u = usr 61324                (no CLS)

9890   let h=-h : let xφ= xφ ±h*cb : let yφ=yφ-h*sb :
       let h1 = h  : let w1 = w
                                                 if h>1 and w>1 then
9900   let q$ = inkey$ ~~if q$ = ""~~
9905   if q$ = "+" then let h=h/2 : let w=w/2 : goto 9998   if h<32768
9910   if q$ = "-" then let h=h*2 : let w=w*2 : goto 9998   and w<32768
                                                            then
                              4*                 4*
9915   if q$ = "8" then let xφ = xφ + dxl : let yφ=yφ + dyl : goto 9643
9920   if q$ = "5" then let xφ = xφ-4*dxl : let yφ=yφ-4*dyl : goto 9643
9925   if q$ = "6" then let xφ = xφ + 4*dxc : let yφ=yφ +4*dyc : goto 9643
9930   if q$ = "7" then let xφ = xφ - 4*dxc: let yφ=yφ-4*dyc : goto 9643
                   or(q$="m")
9935   if(q$ = "M") then ~~goto~~ return
9980   goto 9900
~~9998 let nnx = w/nx : let nny = h/ny : goto 9643~~
9999   return

9998   let xφ = xφ ∓(w ~~*sb/2~~ - w1) * sb/2 + (h-h1) * cb/2 :
       let yφ = yφ ∓(w-w1) * cb/2 ∓ (h-h1) * sb/2 :
       let nnx = w/nx : let nny = h/ny : goto 9643
                                        or((i$="3")and(f$="f"))
9939   if((i$ <> "2") and(i$<>"3")) then goto 9950
9940   if q$ = "T" then  poke 61322,13 : poke 61323, 239 : goto 9643
9945   if q$ = "C" then  poke 61322,187 : poke 61323,238 : goto 9643
9950   if q$ = "D" then goto 9996

       cls:
9996   print at 15,φ; "xφ="; xφ; : print at 16,φ; "yφ="; yφ; :
       print at 15,16; "w=" ; w; "m."; : print at 16,16; "h="; h; :
       print at 17,φ; "x scale = 1: "; int (nnx * 1000); :
       print at 18,φ; "y scale = 1: "; int (nny * 1000); :
       print at 19,φ; "b = "; b; "degrees"; :
       print at 20,φ; "contours"; ci ; "m.";
                  ENTER:
9997   input "~~Any key~~"; q$ : goto 9643                 (10855,4090)
```

The Forest: Main Menu (enhanced).



Memory map for assembly parts of The Forest

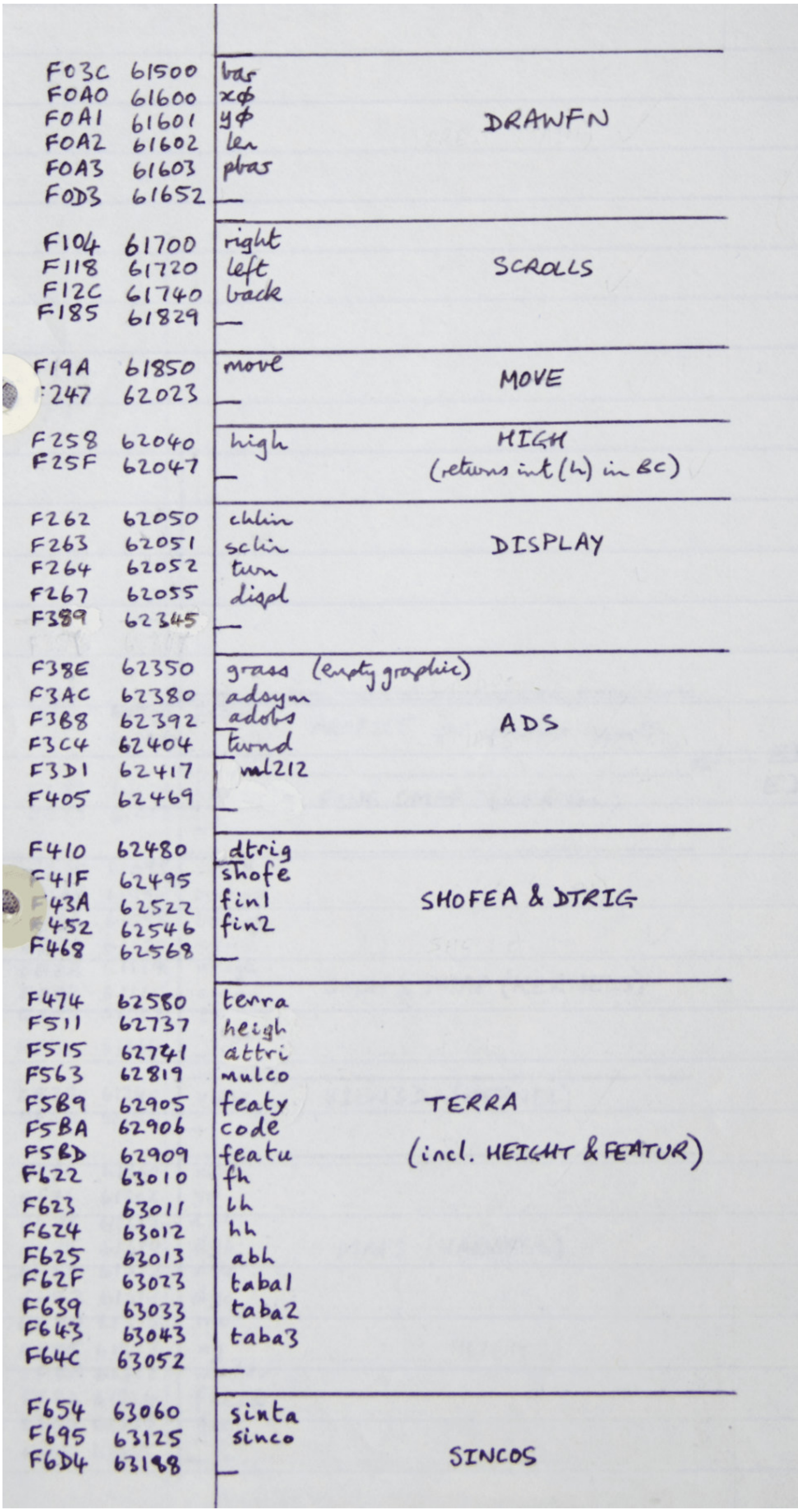
See also page 24.



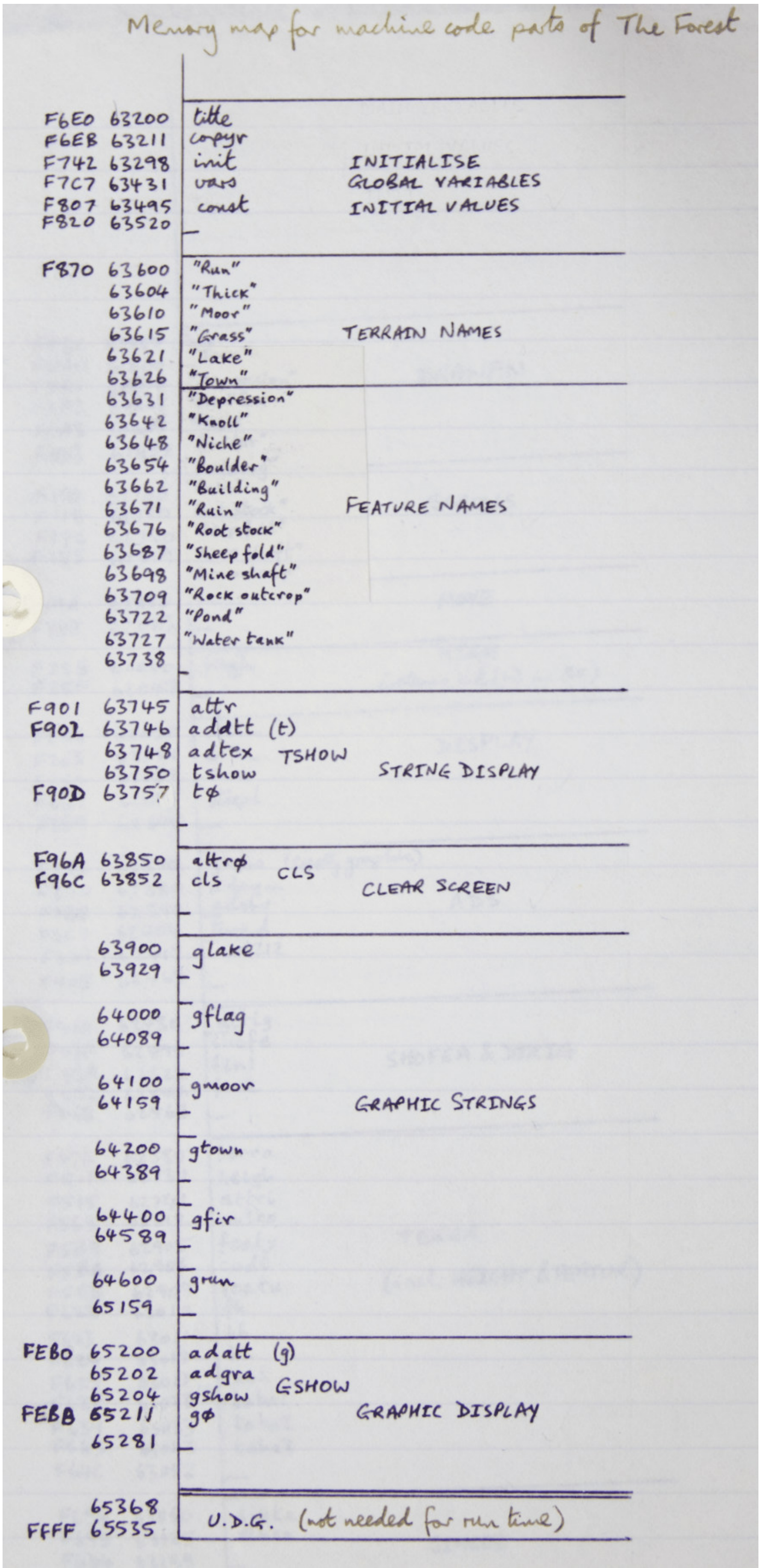
EC54	60500	fstrt	PROFILE for complex forest
	60755		
EE48	61000	finec	FINE CMAP (KERNEL)
EE7D	61053		
EEA0	61088	tend	CMAP & TMAP (KERNELS)
EEA8	61096	tempx	
EEAA	61098	tempy	
EEAC	61100	syms	
EEBA	61114	nrigh	
EEBB	61115	cmap	
EF0D	61197	tmap	
EF34	61236		
EF38	61240	view	VIEW3D (KERNEL)
EF6D	61293		
EF74	61300	xØ	MAPS (HARNESS)
EF76	61302	yØ	
EF7C	61308	dxl	
EF7E	61310	dyl	
EF80	61312	dxc	
EF82	61314	dyc	
EF84	61316	nx	
EF85	61317	ny	
EF8A	61322	maddr	
EF8C	61324	fhead	
EFC9	61385	fret	
F00F	61455		

← ED8Ø 60800
EDBC

TER3D



F03C	61500	bar	DRAWFN
F0A0	61600	x⌀	
F0A1	61601	y⌀	
F0A2	61602	len	
F0A3	61603	pbas	
F0D3	61652	—	
F104	61700	right	SCROLLS
F118	61720	left	
F12C	61740	back	
F185	61829	—	
F19A	61850	move	MOVE
F247	62023	—	
F258	62040	high	HIGH (returns int(h) in BC)
F25F	62047	—	
F262	62050	chlin	DISPLAY
F263	62051	sclin	
F264	62052	turn	
F267	62055	displ	
F389	62345	—	
F38E	62350	grass (empty graphic)	ADS
F3AC	62380	adsym	
F3B8	62392	adobs	
F3C4	62404	turnd	
F3D1	62417	val212	
F405	62469	—	
F410	62480	dtrig	SHOFEA & DTRIG
F41F	62495	shofe	
F43A	62522	fin1	
F452	62546	fin2	
F468	62568	—	
F474	62580	terra	TERRA (incl. HEIGHT & FEATUR)
F511	62737	heigh	
F515	62741	attri	
F563	62819	mulco	
F5B9	62905	featy	
F5BA	62906	code	
F5BD	62909	featu	
F622	63010	fh	
F623	63011	lh	
F624	63012	hh	
F625	63013	abh	
F62F	63023	taba1	
F639	63033	taba2	
F643	63043	taba3	
F64C	63052	—	
F654	63060	sinta	SINCOS
F695	63125	sinco	
F6D4	63188	—	



Memory map for machine code parts of The Forest

F6E0	63200	title		
F6EB	63211	copyr		
F742	63298	init		INITIALISE
F7C7	63431	vars		GLOBAL VARIABLES
F807	63495	const		INITIAL VALUES
F820	63520			
F870	63600	"Run"		
	63604	"Thick"		
	63610	"Moor"		
	63615	"Grass"		TERRAIN NAMES
	63621	"Lake"		
	63626	"Town"		
	63631	"Depression"		
	63642	"Knoll"		
	63648	"Niche"		
	63654	"Boulder"		
	63662	"Building"		
	63671	"Ruin"		FEATURE NAMES
	63676	"Root stock"		
	63687	"Sheep fold"		
	63698	"Mine shaft"		
	63709	"Rock outcrop"		
	63722	"Pond"		
	63727	"Water tank"		
	63738			
F901	63745	attr		
F902	63746	addtt (t)		
	63748	adtex	TSHOW	
	63750	tshow		STRING DISPLAY
F90D	63757	tø		
F96A	63850	attrø		
F96C	63852	cls	CLS	CLEAR SCREEN
	63900	glake		
	63929			
	64000	gflag		
	64089			
	64100	gmoor		
	64159			GRAPHIC STRINGS
	64200	gtown		
	64389			
	64400	gfir		
	64589			
	64600	grun		
	65159			
FEB0	65200	adatt (g)		
	65202	adgra		
	65204	gshow	GSHOW	
FEBB	65211	gø		GRAPHIC DISPLAY
	65281			
	65368			
FFFF	65535	U.D.G. (not needed for run time)		

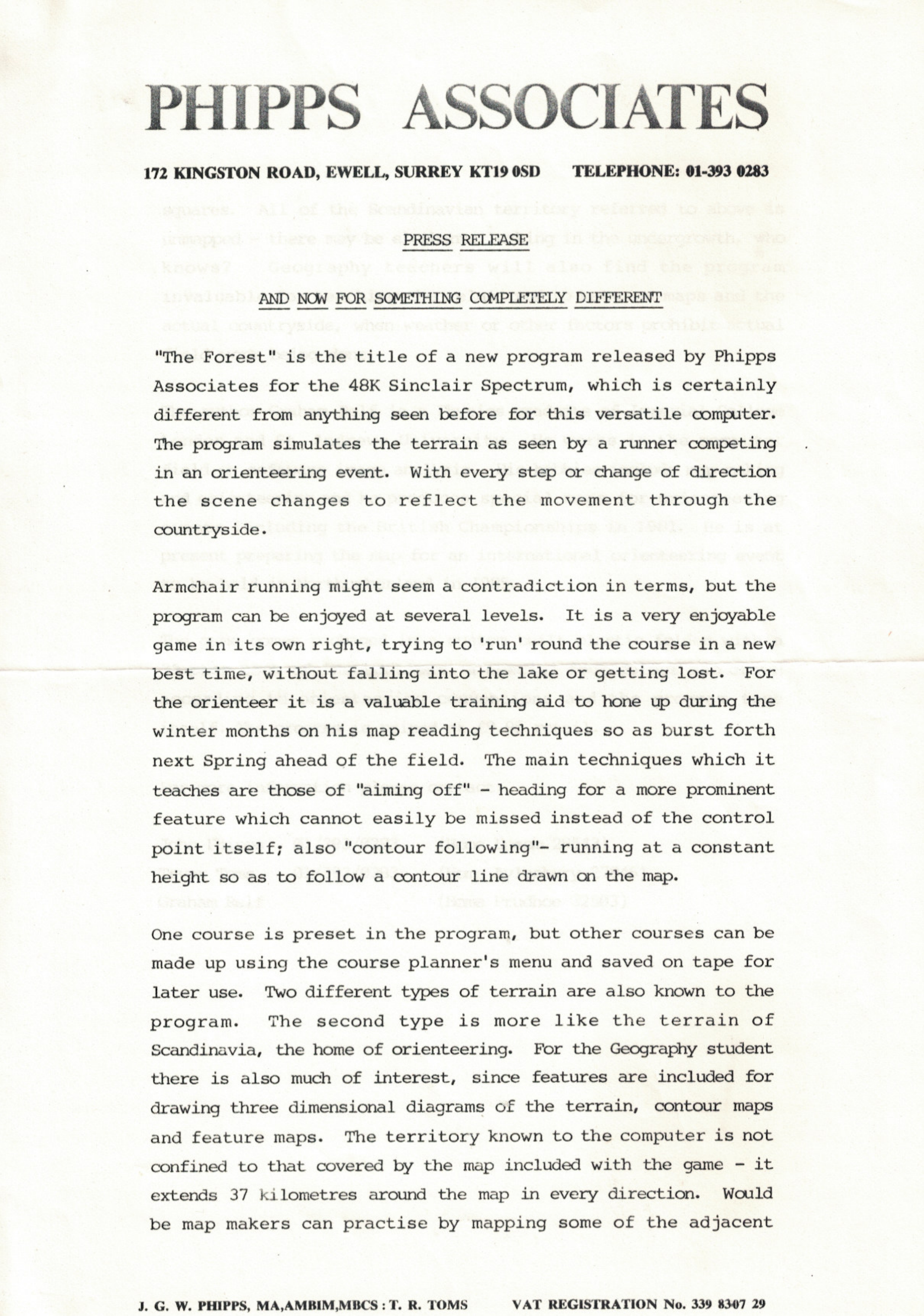
LOONY ZOO Arcade escape game £5.95 PHARAOH'S TOMB Adventure £4.95 KNIGHT'S QUEST Adventure £5.95
COLDITZ Adventure £6.95 THE FOREST Orienteering simulation £9.95

PHIPPS ASSOCIATES

Dept G FREEPOST EM463 (No stamp)
172 Kingston Road, Ewell, Surrey KT19 0BR
Telephone 01-393 0283. 24 Hour answering.

Prices include postage (outside Europe add £1.00 per item). *Access and Visa cards welcome*

VISA



PHIPPS ASSOCIATES

172 KINGSTON ROAD, EWELL, SURREY KT19 0SD TELEPHONE: 01-393 0283

PRESS RELEASE

AND NOW FOR SOMETHING COMPLETELY DIFFERENT

"The Forest" is the title of a new program released by Phipps Associates for the 48K Sinclair Spectrum, which is certainly different from anything seen before for this versatile computer. The program simulates the terrain as seen by a runner competing in an orienteering event. With every step or change of direction the scene changes to reflect the movement through the countryside.

Armchair running might seem a contradiction in terms, but the program can be enjoyed at several levels. It is a very enjoyable game in its own right, trying to 'run' round the course in a new best time, without falling into the lake or getting lost. For the orienteer it is a valuable training aid to hone up during the winter months on his map reading techniques so as burst forth next Spring ahead of the field. The main techniques which it teaches are those of "aiming off" - heading for a more prominent feature which cannot easily be missed instead of the control point itself; also "contour following"- running at a constant height so as to follow a contour line drawn on the map.

One course is preset in the program, but other courses can be made up using the course planner's menu and saved on tape for later use. Two different types of terrain are also known to the program. The second type is more like the terrain of Scandinavia, the home of orienteering. For the Geography student there is also much of interest, since features are included for drawing three dimensional diagrams of the terrain, contour maps and feature maps. The territory known to the computer is not confined to that covered by the map included with the game - it extends 37 kilometres around the map in every direction. Would be map makers can practise by mapping some of the adjacent squares. All of the Scandinavian territory referred to above is unmapped - there may be elephants lurking in the undergrowth, who knows? Geography teachers will also find the program invaluable for teaching the relationship between maps and the actual countryside, when weather or other factors prohibit actual field work being done.

The author Graham Relf is a Physics graduate of Imperial College London and St. Andrews University. He works in the computer field on software image analysis. His hobbies include map making and orienteering and he produces special maps for orienteering events including the British Championships in 1981. He is at present preparing the map for an international orienteering event to be held in Northumberland in 1985.

The game comes packaged in a custom built plastic folder with a 32 page book of instructions, a special four colour map drawn according to orienteering conventions and the program tape itself. The program is priced at £9.95 retail.

For more information please contact:

John Phipps	Ø1/393/Ø283	(Home Epsom 28543)
Trvor Toms	Ø1/393/Ø283	(Home Aylesbury 27846)
Graham Relf		(Home Prudhoe 325Ø3)

Bob Chappell takes a crash course in orienteering in the comfort of his own front room.

Into the Forest

I was sure it had nothing to do with a certain East London soccer team. Maps were involved somewhere along the line, yet more than that I could not say. The ignorance is now resolved and I am a changed man. I have been through The Forest.

As you're probably wondering what on earth I'm on about, let me explain in one word — orienteering. The instrument of my education was a program called, unassumingly, The Forest. If you know as little about orienteering as I did, then this program is probably the best practical introduction to the sport you could find.

The Forest is a simulation of the sport of orienteering. With it you can travel miles through the toughest terrain without ever leaving your fireside chair. Orienteering can most simply be described as car rallying on foot, being a cross-country race over difficult terrain, often forest, with the extra dimension of having to navigate your own course.

Armed with a special marked map and a compass, you have to plot and follow your route between a number of control points from the start to the finish. At each control point, a control card is marked as a verification that all the correct checkpoints have been visited. The art lies in successful map reading as much as being fast on your feet.

The program is intended as a training aid for practising the techniques of orienteering and can be used in teaching the understanding of maps and their relationship to the physical world. It also happens to be a very entertaining and demanding sport simulation in its own right and will undoubtedly appeal to those games-players who like their pleasures to be more cerebral. Tackled as a race, yomp or steady stroll, it offers many interesting challenges during the course and much satisfaction on successful completion.

A detailed map of the course is supplied with the package. The screen views the terrain directly ahead. All visible features (trees, towns, lakes, etc) are shown on the horizon which is six metres away. The view covers a 100 degree sector, 50 degrees to each side of the centre. Movement through the countryside is controlled by the cursor keys. Pressing the left or right arrow key faces you 11.5 degrees to the left or right respectively. Pressing the down arrow causes a 180 degree about-face. The up arrow sets you moving forward, each step being about 1 metre. To stop running press any key except the up arrow.

The direction you are facing can also be changed by altering your bearings; these can be taken and changed at any time. With every change of direction, the display is updated accordingly. A list of the control point codes and descriptions can be displayed, as can your control card which may be punched at any control point. The time elapsed since you started off is displayed on request.

Assistance can be obtained by calling up a menu and getting your current co-ordinates, enabling you to verify your position against the map. Any part of the terrain can be displayed as terrain type, as contours, as a cross-section, as a three-dimensional view, or can be scanned for point features. These features are not there just to get you out of a jam but are intended to instruct by showing the relationship between the map and the ground in a variety of ways. These features are most impressive and geography teachers could find them invaluable as a teaching aid.

You can give up before you've finished — the display tells you which of the control points marked on your card are correct or missing. There is an option to try out the same course on a more complex forest, which is considerably more difficult.

One might think that having completed the course once or twice the challenge would disappear. Not so, for the author has provided a facility to set out your own course over the mapped area. There are a large number of control points laid out in the terrain, only a few of which are used in the preset course. By following a simple question and answer menu, you may use any of these control points to construct your own course. The only restriction is that the start and finish points must be those used in the preset course.

Being able to design then save these courses to tape extends the interest value of the program indefinitely. With the additional element of competing against the clock and the various facets of the mapping aids, the program offers vast scope for entertainment and instruction.

Presentation

The whole package has the hallmarks of quality and professionalism. The program comes on cassette packed in a large video-style case. The program is copied to both sides of the cassette and it loaded first time, every time. Inside the case is the printed map. It is very well produced and the scale and legends are clearly marked, as is the course.

The 32 page book that comes with it provides an introduction to orienteering and covers all the features of the program clearly and concisely. There are helpful diagrams complementing the text which assist in understanding trickier concepts such as map orientation for three-dimensional displays.

Although I knew precious little about orienteering before I picked up this book, I found that I could quickly become proficient in using the program effectively and, more to the point, understand precisely what was going on and why. The book is lacking in two places only — in setting up your own course and in restarting the program. Apart from that, the documentation, like the program, is first-rate.

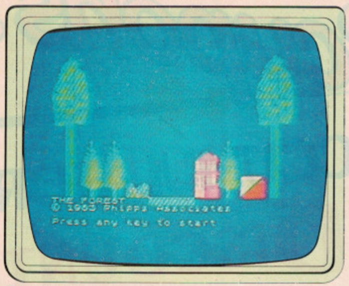
Getting started

Although the program is very easy to operate, you are well advised to study the manual before plunging into The Forest. Of particular importance is the recommendation to use a protractor for taking bearings on your map. The manual gives invaluable advice on navigational techniques — following boundaries, counting paces, lining up on landmarks, checking bearings, etc. I ignored this expertise to start with and, although I had no problems in using the program and made considerable progress across country, I ended up way off course.

On running the program, you are immediately presented with a colourful view of trees and part of a town. This is

35 ▶

Johanne Ryder



THE FOREST
© 1983 Phipps Associates
Press any key to start

The starting point.

◄ 33

the starting location. Typing shifted B shows your current bearing and invites you to enter a new one. Having done this, pressing the upward arrow sets you off at a steady pace in the direction selected by your bearing. The scene is quickly updated with every step taken.

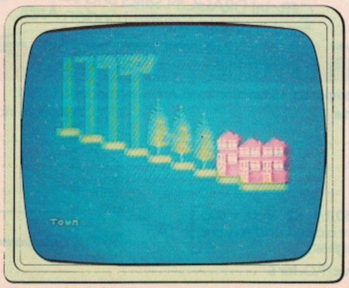
In real life, you would be almost certain to drift slightly from your set course, and since the program emulates this tendency it is worthwhile checking your bearings at regular intervals. Failure to do this had me missing the first control point and having to retrace my steps. Pressing any key other than the upward arrow causes you, and the displayed scene, to remain motionless — but time marches on. Pressing the left or right arrow key while moving has the scene sliding smoothly sideways in the desired direction.

The manual clearly explains the displayed symbols and offers advice on navigating for the control points. With this guidance, I found it very easy to get started and soon had the first three of the 12 control points stamped on my card. It was not long after that that I became hopelessly lost and had to resort to 'cheating' by calling up the assistance menu. This put me back on the right track, although I had wasted a lot of time looking for the control flag in the wrong place.

In use

The graphic display, with its representations of thicket, houses, moor, grass, lakes and buildings, gives a clear impression of the terrain before you. Eye level is represented by a central square and, as the scene varies in height, it is easy to determine whether you are going uphill or down. Colours are used naturally and effectively.

The contour map option allows you to select a fine or coarse map of an area. If the latter, the contour interval can be chosen and the resulting map coloured and lettered (the higher the letter in the alphabet, the higher the contour). Maps can be scaled or drawn covering a given width and height. The orientation of the map can also be specified.

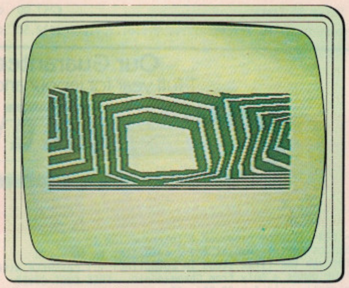


The result of sideways movement.

Mapping the terrain gives a coloured and lettered representation of the area, each letter repesenting a type of terrain (*eg* g for grass, m for moor, etc).

The point features option allows the scanning of a particular area. The display is exactly how you see it when in the normal mode of traversing the countryside. The scan runs up and down the area in raster scan fashion, covering a strip at a time, just like a lawn mower. The scan displays the co-ordinates at every step and halts at any section containing any point feature (knolls, depressions, boulders, etc) within it — useful for checking out the names and locations of the control flags.

A cross-sectional view of an area can be taken and a three-dimensional view of a particular area can be obtained. In the latter, being able to specify the orientation of the view is especially useful.



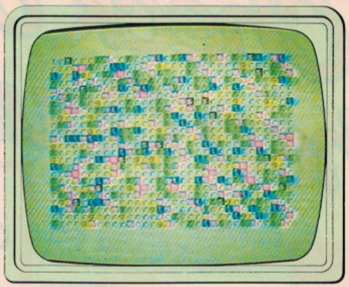
The ordinary contour map.

Setting up one's own course is simple, but you have to know the names and associated descriptions of the control flags you are going to use. Since these can only be found by exploring the terrain (they are not marked on the map or listed in the manual — that would spoil the fun), you need to build up this list yourself. You can't enter new flags or descriptions. Co-ordinates of the control points for your own course do not need to be specified, only the flag codes and associated descriptions. You can have up to 20 control points on your course.

Reliability

The single key commands and menu of options make the program easy to use. When you have completed a course, your card is displayed along with the time, markings and number of times you cheated. At this point the program is halted by a STOP statement. If you wish to have another race, the manual tells you to type in GOTO 4.

The program loads in two sections — what the manual doesn't say is that you should rewind the tape to the beginning of section two (or even to the beginning of the reel) before executing the GOTO otherwise the program will hang, not being able to find that section to load. One might just as easily power off and on, and reload from scratch.



A full colour map of the terrain.

Punching your control card with the wrong flag code or against the wrong control point is possible, but then this would be possible if you were really orienteering. It is necessary to use the D option to check the control codes and descriptions before punching your card.

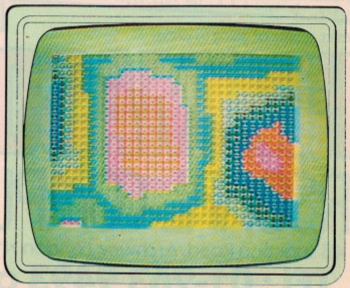
Don't think you can cheat by learning the codes for each of the course points, punching them out on your card and going straight to the finish. The program won't let you punch a control on your card unless you are close by the flag in question.

Verdict

An exceptionally absorbing program and one that is certain to sustain one's interest over a long period of use. If you are looking out for a program that is original, entertaining, instructive, intellectually satisfying and has an unlimited interest span, then I unreservedly recommend The Forest to you. I just hope that this is but the first in a long series of orienteering programs from Mr Relf and Phipps Associates.

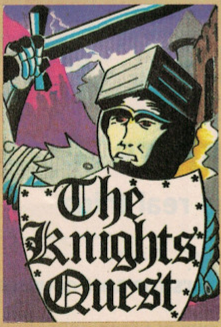
Name The Forest **System** Spectrum (48K) **Price** £9.95 **Publisher** Phipps Associates, 172 Kingston Road, Ewell, Surrey KT19 0SD Tel 01-393 0283 **Format** Cassette **Language** Basic and machine code **Other versions** None **Outlets** Mail order and dealers

RATING	
Features	▣▣▣▣
Documentation	▣▣▣▣
Performance	▣▣▣▣
Usability	▣▣▣▣▣
Reliability	▣▣▣▣
Overall value	▣▣▣▣▣

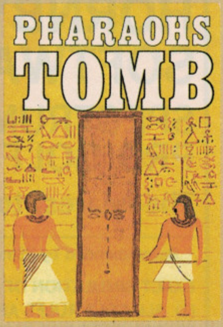


Contour map showing relative heights.

48K ZX SPECTRUM ADVENTURES – PICTURE GRAPHICS AND COLOUR



You are a Knight of Camelot, searching for Merlin's lost treasure. On your way you will discover the Witches' Tower, rescue a Princess held by the wicked Wizard of Trill. **£5.95**

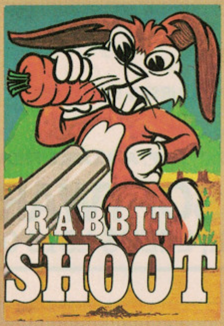


You discover the entrance to an ancient pyramid blocked by a rock. Once inside, you discover fire rooms, ice rooms and other traps set by the builders to protect the Pharaoh. **£4.95**

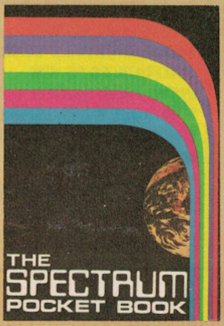
A rope above a rock fissure is the only way into this Magic Mountain, or is it? Legends tell of vast stores of treasure but also of poisonous spiders, lizards and magic at work. **£4.95**

An old deserted mining town holds the clues to the location of a lost gold mine. Once in the mine, your problems are not over – the roof creaks alarmingly and might cave in. **£4.95**

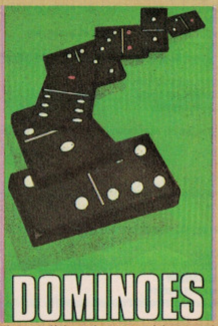
16K ZX SPECTRUM GAMES AND PUZZLES



It is nearly dawn, you are poaching rabbits in a field of carrots. See how many you can bag undetected by the game keeper. Highly original arcade style game. **£4.95**



The cassette of the book. Contains six games (including Castle Walls, Great Fire of London, Reversi) machine code assembler, disassembler. **£5.95**
Book available separately **£6.50**



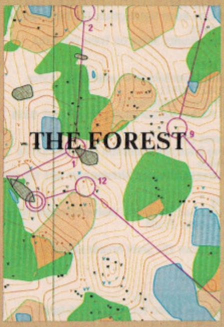
The traditional game with superb screen presentation. Score points by making the two ends add to a multiple of five or three. The first one to reach 72 points wins. **£4.95**

The computer breaks and shuffles a two by two coloured square, whilst you watch the moves it makes. You must then unscramble it to reassemble the original squares. Machine coded. **£4.95**

48K ZX SPECTRUM GAMES

To find the Black Planet you need 7 pieces of the key each hidden on different planets, and needing different puzzles to be solved. On the way, you fight off the pirates who get steadily more desperate. **£5.95**

You are Spectrasses, battling in the arena with Gorgon, whose stare can turn you to stone. To recover the lost chalice, you must also win a swordfight with Grang, inside his cave. **£4.95**



Three dimensional simulation of the sport of orienteering. Display is continuously updated as you run. Ideal for map reading practice. Instructions, colour map and cassette. **£9.95**

16K ZX81

For the ZX81, three adventures – black and white text. Magic Mountain, Pharaoh's Tomb, Greedy Gulch, as described above. Superb value for money. **£5.00**

PHIPPS ASSOCIATES

Dept G [illegible] T EM463 (No stamp)
172 K[illegible] Ewell, Surrey KT19 0BR
Telephone [illegible] 3 0283. 24 Hour answering.

Prices include postage (outside Europe add £1.00 per item). *Access a[illegible] cards welcome*

VISA

Your Spectrum, June 84

FRONTLINES

FORESTRY COMMISSION

Calling all *Forest* devotees, Have you been amazed to discover that yo never run off the edge of the action although *your* map is only 2Km by 2Km? Then grab pen and paper and immediately send off for your *Complex Forest* map from Phipps Associates, 172 Kingston Road, Ewell, Surrey (or phone 01-393 0283).

NEW 'FACE FOR THE SPECCY

Ram Electronics, the Fleet-based hardware company, recently announced the launch of a new 'multi-purpose' Spectrum interface.

The device, modestly named the 'Ram Turbo', accepts Sinclair ROM cartridges and two joysticks of the standard 9-pin 'D' connector type. At this point you may be thinking this sounds rather like one of Sinclair's own Speccy add-ons — and you'd be right.

Although the Ram Turbo is £3 more expensive than the ZX Interface 2, it looks more futuristic and has a couple of extra design features. A unique built-in safety device prevents the user from inflicting expensive damage on the Speccy — the power cable can only be connected once the interface is in place. The Turbo also offers a full expansion bus (unlike Interface 2) which means there are no restrictions as to what other bits you may be thinking of bolting on the back.

The futuristic Ram Turbo incorporates a unique built-in safety device.

"We expect to sell over 50,000 Turbos in the first three months alone," says technical director Martin Shoebridge. "Technically it's miles ahead of any competitors (surely not a dig at Uncle Sir C!) but most people want it yesterday — not next month."

We trust this isn't just the latest contender for the Flying Pigs department. More information, etc., from Ram Electronics on 02514 5858.

PARK IT IN THE ROM

Parker has moved into the home micro market with

ON THE CARDS 1

HOBBIT HACKERS

For all of you who bought *The Hobbit*, avidly read the book, tried to play the game and were *still* confused, Melbourne House has

Further information

The Forest on the World of Spectrum web site.

A 1984 review of the original version, in Crash magazine.